UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE AUTOMAÇÃO E SISTEMAS

William Andrey Faustino Scotti

ARQUITETURA DE SISTEMA DE CONTROLE SUPERVISÓRIO INTEGRANDO CLP, SCADA E ROTEAMENTO DE TAREFAS

Florianópolis

2015

William Andrey Faustino Scotti

ARQUITETURA DE SISTEMA DE CONTROLE SUPERVISÓRIO INTEGRANDO CLP, SCADA E ROTEAMENTO DE TAREFAS

Dissertação submetida ao Programa de Pós-Graduação em Engenharia de Automação e Sistemas para a obtenção do Grau de Mestre.
Orientador: Prof. Max Hering de Queiroz, Dr.
Co-orientador: Prof. José Eduardo Ribeiro Cury, Dr.

Florianópolis

2015

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Scotti, William Andrey Faustino
Arquitetura de sistema de controle supervisório integrando CLP, SCADA e roteamento de tarefas / William Andrey Faustino Scotti ; orientador, Max Hering de Queiroz ; coorientador, José Eduardo Ribeiro Cury. - Florianópolis, SC, 2015.
116 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Engenharia de Automação e Sistemas.

Inclui referências

1. Engenharia de Automação e Sistemas. 2. Sistema de eventos discretos. 3. Teoria de controle supervisório. 4. Controlador lógico programável. 5. Sistema de roteamento de tarefas. I. Queiroz, Max Hering de. II. Cury, José Eduardo Ribeiro. III. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia de Automação e Sistemas. IV. Título.

William Andrey Faustino Scotti

ARQUITETURA DE SISTEMA DE CONTROLE SUPERVISÓRIO INTEGRANDO CLP, SCADA E ROTEAMENTO DE TAREFAS

Esta Dissertação foi julgada aprovada para a obtenção do Título de "Mestre", e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia de Automação e Sistemas.

Florianópolis, 27 de Março 2015.

Prof. Rômulo Silva de Oliveira, Dr.
Coordenador

Prof. Max Hering de Queiroz, Dr.
Orientador

Prof. José Eduardo Ribeiro Cury, Dr.
Co-orientador

Banca Examinadora:

Prof. Victor Juliano De Negri, Dr.

Prof. Marcelo Ricardo Stemmer, Dr.

Prof. Agnelo Denis Vieira, Dr.

Dedico este trabalho à toda a minha Família, em especial à memória do meu avô, Aldo Faustino.

AGRADECIMENTOS

Agradeço em primeiro lugar a Deus que me guiou nessa longa caminhada, e me ajudou a superar todas as etapas necessárias até concretização desta pesquisa. Agradeço também:

À Universidade Federal de Santa Catarina, em especial ao Programa de Pós-Graduação em Engenharia de Automação e Sistemas e todo o seu corpo docente, pela oportunidade de atuar como aluno.

À CAPES pela concessão da bolsa de estudo que viabilizou o pleno desenvolvimento de meu Mestrado.

Ao meu orientador, professor Max Hering de Queiroz pela amizade e companheirismo, mas principalmente pela confiança a mim atribuída. Obrigado por ter acreditado em mim em todos os momentos.

Ao meu co-orientador Jose Eduardo Ribeiro Cury pelo incentivo e valiosas contribuições sugeridas para enriquecer o trabalho apresentado.

Aos amigos da turma de Mestrado, em especial a Fernando Silvano, Cleber Pinelli e o Rafael Toledo, por todo o apoio recebido, por todas as dificuldades superadas, por todos os bons momentos vividos e principalmente pela amizade que levo junto.

A todos os meus amigos, em especial a Adir Trevisan Junior, pela amizade de longa data, pelo companheirismo e apoio em todas as horas. Agradeço também ao meu amigo e colega de graduação Bruno Vitória, por toda amizade e bons momentos.

À empresa Greylogix Brasil, em especial a Rafael Gonçalves d'Ávila da Silva, que em todos os momentos me ajudou e incentivou na conclusão dessa dissertação.

À toda minha Família maravilhosa: Pais, irmão, tios e avós, por todo amor e carinho, que mesmo longe, me apoiaram e ajudaram em todos os momentos.

Ao meu irmão, Kelvin Scotti, que em todos os momentos da minha vida esteve sempre ao meu lado. Deixo registrado aqui o meu amor de irmão que sempre estará ao seu lado.

E por fim, a minhã mãe, Naricleia Terezinha Faustino, por todo amor, carinho, paciência e acima de tudo por sempre me apoiar e incentivar em todas as minhas escolhas. Amo eternamente você!

Enfim, agradeço a todos que de uma forma ou de outra contribuíram para o desenvolvimento dessa Dissertação.

A melhor coisa sobre o futuro é que vem um só dia de cada vez

(Abraham Lincoln, 1809-1865)

RESUMO

Este trabalho desenvolve uma metodologia para a síntese e implementação do sistema de controle supervisório para sistemas de manufatura, integrando o programa de Controlador Lógico Programável (CLP), o SCADA e o roteamento de tarefas através da Teoria de Controle Supervisório (TCS). A abordagem modular local da TCS é utilizada para a modelagem da síntese formal e simulação de uma lógica de controle minimamente restritiva, representada através de um conjunto de autômatos de tamanho reduzido. Esses autômatos são usados tanto para a geração automática de código de CLP, quanto para a estruturação das diversas funcionalidades do SCADA. Um sistema complementar faz uso de heurísticas para a escolha do próximo comando a executar dentre aqueles habilitados pelo controle supervisório, buscando otimizar o roteamento de tarefas segundo algum critério de produção. A aplicação da metodologia a um sistema modular de produção didático mostrou que a abordagem proposta beneficia a sistematização, modularidade e estruturação do projeto do sistema de controle supervisório, assim como a eficiência, segurança e flexibilidade do sistema de manufatura.

Palavras-chave: Teoria de Controle Supervisório. Sistema de Manufatura. Sistema de Controle Supervisório. Controlador Lógico Programável. SCADA. Roteamento de Tarefas.

ABSTRACT

This work aims to develop a methodology to synthesize and implement a supervisory control of manufacturing systems, by integrating a programmable logic controller (PLC), the SCADA (supervisory control and data acquisition) and the tasks routing by utilizing the Supervisory Control Theory (SCT). The local modular approach of TCS is used to model the formal synthesis and simulation of a minimally restrictive control logic, represented by a set of size reduced automatons. These Automatons are used either to automatically generate a PLC code or to structurate several functionalities of SCADA. A complementary system utilizes heuristics to decide the next command to be executed among those enabled by supervisory control, in order to optimize the tasks routing according to production criterion. The aplication of this methodology to a didatic modular production system shows that the proposed approach benefits the systematization, modularity and structure of the supervisory control system design, as well as it reinforces the efficiency, safety and flexibility of the manufacturing system.

Keywords: Supervisory Control Theory. Manufacturing System. Supervisory Control System. Programmable Logic Controller. SCADA. Task Routing.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SUMÁRIO

1 INTRODUCAO

Nos tempos atuais, as exigências por ciclos de produção menores, demanda de produção imprevisível e maior variedade de produtos faz com que as empresas busquem métodos não tradicionais de fabricação, trazendo à tona a necessidade de estudos de novos conceitos. Para atender essa necessidade, Rogers e Bottaci (1997) propuseram um novo paradigma de produção: Sistemas Modulares de Produção (*Modular production systems* - MPS). Tais sistemas são compostos por máquinas modulares e controladores reconfiguráveis, em um conceito que combina a alta taxa de transferência de linhas de produção dedicadas com a flexibilidade de Sistemas Flexíveis de Manufatura (FMS). O projeto modular de um MPS permite a diversidade da produção por meio de diferentes combinações de máquinas em um processo, a fim de se ajustar rapidamente a capacidade de produção e funcionalidade. O conceito MPS conta com um design modular e sistemático para a reconfiguração do layout de produção, do hardware da máquina e do sistema de controle.

Um exemplo de aplicação do sistema modular é visto na indústria automotiva. Na fase final de montagem, a montadora tem a oportunidade de realizar diferentes combinações de acabamentos (bancos, sistema de combustível, painéis, etc.) atendendo diferentes mercados, como por exemplo montar veículos com direção à esquerda ou à direita.

Koren et al. (1999) apresentam o conceito de Sistema Reconfigurável de Manufatura (*Reconfigurable Manufacturing Systems* - RMS) semelhante ao MPS. A ideia é que os sistemas são fabricados desde o início com recursos ajustáveis para proporcionar exatamente a capacidade e as funcionalidades que são necessárias, adaptando-se às condições do mercado rapidamente.

Esses sistemas agregam tecnologias modernas de automação, que visam integrar ampla gama de novas tecnologias de processo produtivo, trazendo benefícios econômicos e sociais. Com a crescente evolução da automação industrial e devido às muitas tarefas da automação, uma subdivisão das tarefas é necessária. Moraes e Castrucci (2007) classificam a automação em cinco níveis hierárquicos, que vão desde os equipamentos e dispositivos em campo até o gerenciamento corporativo da empresa. Essa subdivisão leva em conta a ISA-95 (2010), a qual é a mais importante publicação sobre integração de sistemas industrias e se tornou a referência no desenvolvimento destas soluções. A subdivisão é vista na Figura 1.

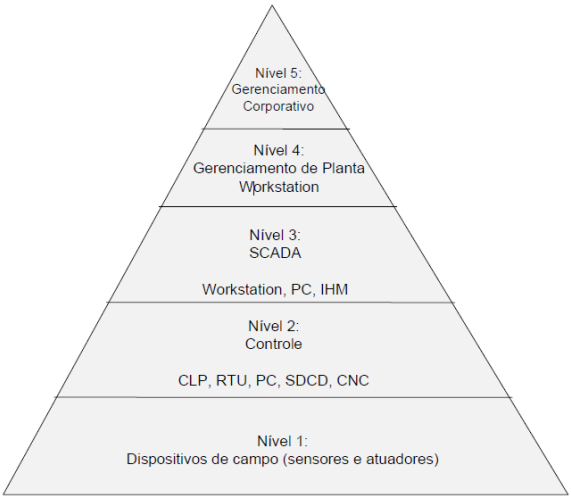

Figura 1 – Pirâmide da Automação.
Fonte: Adaptado de (MORAES; CASTRUCCI, 2007).

O nível 1, chamado de "chão de fábrica" é o nível onde se encontram as máquinas e os dispositivos de campo, como atuadores, sensores, transmissores e outros. O nível 2 é onde se encontram concentradas as informações sobre o nível anterior e engloba dispositivos que realizam o controle automatizado das atividades da planta, como Controladores Lógicos Programáveis (CLPs), Sistemas Digitais de Controle Distribuído (SDCDs), Unidades Terminais Remotas (*Remote Terminal Units* - RTUs), Dispositivos Eletrônicos Inteligentes (*Intelligent Electronic Device* - IEDs), entre outros. No nível 3 encontram-se os sistemas SCADA (*Supervisory Control and Data Acquisition*), os quais concentram as informações passadas pelos equipamentos dos níveis 1 e 2 em banco de dados e as repassam para os níveis administrativos (níveis 4 e 5), além de prover diversas funcionalidades para supervisão da planta. O nível 4 é responsável pela programação e planejamento da planta fabril, realizando o controle e a logística dos suprimentos e definindo as tarefas para o nível 3. É também nesse nível que se encontra o Sistema de Execução da Manufatura (Manufacturing Execution System - MES), responsável por monitorar a produtividade da indústria em tempo real. O nível 5 é responsável pela administração de todos os recursos da empresa. Neste nível encontram-se *softwares* para gestão de venda, gestão financeira e Inteligência empresarial (*Business*

Intelligence - BI) para ajudar na tomada de decisões que afetam a empresa como um todo (MORAES; CASTRUCCI, 2007). É nesse nível que o Planejamento e Controle da Produção (PCP) é realizado.

Com essa hierarquia, pode-se ter uma organização maior dos diferentes níveis de controle existentes. Nos níveis mais baixos encontram-se os equipamentos que estão diretamente relacionados à utilização em campo, já nos níveis mais altos são tratadas questões mais gerenciais da empresa e da planta. Este trabalho tem foco principal nos níveis 2, 3 e 4 da pirâmide da automação, utilizando um CLP como controlador no nível 2 e o SCADA como supervisor do processo no nível 3. No quarto nível é tratado parte do MES responsável pelo Roteamento de Tarefas da planta a partir de especificações de produção do nível 5.

Equipamentos como CLPs são responsáveis por controlar a planta em tempo real e repassar os comandos para os atuadores da planta. Eles podem operar de forma isolada ou de forma integrada, conectados em rede entre si com um sistema supervisório. O crescimento do CLP e de suas funcionalidades permitiu a sua utilização em indústrias de processo e a conexão com um sistema SCADA (ROSARIO, 2009).

Segundo Silva e Salvador (2011), sistemas SCADA coletam informações de um processo por meio de equipamentos de aquisição de dados, para serem manipuladas, analisadas, armazenadas e apresentadas ao usuário. O sistema SCADA é usado principalmente para iniciar e parar unidades remotas, controlar e monitorar processos que tenham muitas operações de liga e desliga e com poucas malhas de controle analógico PID (Proporcional Integral Derivativo) (RIBEIRO, 2005).

Os equipamentos básicos de sistemas SCADA são:

- CLP para fazer a coleta dos dados;
- Computador Pessoal (PC) para rodar o SCADA e constituir a estação de operação ou a Interface Humano Máquina (IHM) (RIBEIRO, 2005).

Sistemas SCADA atuam em uma variedade enorme de processos, podendo ser encontrados em áreas industriais (celulose, petróleo, metalurgia, química, etc), no controle ferroviário e aeroviário, e em redes de computadores comerciais e residenciais (PINHEIRO, 2006). Assim, plantas de produção automatizadas são controladas por sistemas de controle supervisório, que são responsáveis pelo acompanhamento e execução adequada da produção. Os sistemas supervisórios são geralmente implementados através de um CLP, que comanda as ações de acordo com o comportamento observado a fim de manter a segurança e o sequenciamento do sistema, e são coordenados por um sistema SCADA.

A programação do CLP e a implementação de SCADA são muitas vezes desenvolvidas com base na experiência do programador, sem uma arquitetura estruturada. Embora a literatura acadêmica compreenda muitos usos de métodos formais para projeto do código CLP, a sua aplicação em projetos de sistemas SCADA ainda é uma questão em aberto (PORTILLA, 2011).

Para fins de projeto com lógica de controle, sistemas de manufatura podem ser classificados como Sistemas a Eventos Discretos (SED) que são sistemas dinâmicos cuja mudança de estado ocorre em pontos discretos do tempo, em decorrência de eventos isolados (CASSANDRAS; LAFORTUNE, 2008). Entre os vários métodos formais de controle de SED, a Teoria de Controle Supervisório (TCS) (RAMADGE; WONHAM, 1989) e (WONHAM, 2008) fornece uma estrutura que permite a síntese automática de controle ótimo não-bloqueante, baseada em modelos de autômatos. A TCS determina que eventos que afetam o comportamento de um SED (planta) são inibidos por um agente de controle, chamado supervisor, de acordo com as especificações definidas. A ação do supervisor é calculada para ser não-bloqueante e minimamente restritiva, de modo a inibir apenas os eventos que não estão em conformidade com as especificações, permitindo a execução de todos aqueles que estão em conformidade. No entanto, quando se trata de problemas complexos, a síntese de supervisores enfrenta alto custo computacional e sua implementação no CLP é difícil, uma vez que o número de estados de autômatos cresce exponencialmente com a composição dos subsistemas e especificações.

A abordagem modular local da TCS (QUEIROZ; CURY, 2000) permite explorar a modularidade das especificações e da planta, a fim de obter vários supervisores modulares que agem localmente, além de explorar a estrutura naturalmente descentralizada dos sistemas de manufatura automatizados possibilitando um esforço computacional menor. A complexidade computacional da síntese e o tamanho dos supervisores locais são reduzidos, o que facilita a implementação do sistema de controle. A Arquitetura Modular de Controle Supervisório (AMCS) proposta por Queiroz e Cury (2002b) organiza a implementação de supervisores modulares locais em três níveis hierárquicos, o que pode ser traduzido em código estruturado para CLP de acordo com IEC61131-3 (INTERNATIONAL..., 1998) (PORTILLA; QUEIROZ; CURY, 2014).

Para integrar o desenvolvimento de sistemas SCADA com a programação de controle supervisório em CLP em um sistema de manufatura, Portilla (2011) desenvolveu uma metodologia que é composta de oito fases: projeto informacional; síntese de controle supervisório;

emulação da atuação dos supervisores na planta; implementação do controle supervisório em CLP; implementação de funcionalidades básicas do sistema SCADA; avaliação de funcionamento do sistema real; implementação de funcionalidades gerais do sistema SCADA e, validação do sistema integrado.

A utilização dessa metodologia traz vantagens tanto para o desenvolvimento e aplicação do controle supervisório em CLP quanto para a estruturação de sistemas SCADA. A metodologia mostrou mais flexibilidade e eficiência ao projeto de controle e supervisão de sistemas de manufatura, além de ter uma maior estruturação ao projeto de controladores, deixando a lógica de controle e sua programação mais clara para o projetista e os operadores do sistema (PORTILLA, 2011). A estrutura de controle proposta habilita de forma minimamente restritiva as sequências de eventos que podem ser executados de modo seguro e não-bloqueante pelo PCP. No entanto, Portilla (2011) não especifica um método eficiente para a escolha de qual comando deve ser executado pelo CLP dentre aqueles habilitados.

Este trabalho de dissertação propõe realizar a escolha de comandos do controle supervisório a partir de um Sistema de Roteamento de Tarefas (SRT) responsável pelas decisões de alocação e liberação de recursos do processo, de acordo com informações fornecidas pelo sistema SCADA e PCP. Assim, esse trabalho tem como objetivo desenvolver um método para integrar o sistema SCADA com a implementação de controle supervisório para sistemas de manufatura em um CLP conjuntamente com a definição de um método para executar o SRT em um MPS. O método é aplicado em um módulo de MPS didático para ilustrar e validar a proposta.

Na literatura é encontrado um trabalho relacionado com o roteamento de tarefas em um sistema de manufatura (MOLINA, 2007), o qual utiliza uma estrutura híbrida (TCS e Redes de Petri) para o controle do sistema. Outro trabalho que trata sobre roteamento, é apresentado por Silva et al. (2011), onde os autores lidam com o problema de roteamento através de uma aplicação em uma célula de manufatura real.

Este trabalho está dividido em 6 capítulos. O próximo capítulo traz uma fundamentação teórica acerca de controle supervisório. O terceiro capítulo apresenta o trabalho desenvolvido por Portilla (2011) ao mesmo tempo em que aplica a metodologia a um Sistema Modular de produção. O Capítulo 4 traz as contribuições desse trabalho, principalmente em relação ao desenvolvimento de um Sistema de Roteamento de Tarefas, que é integrado à metodologia desenvolvida no Capítulo 3. Ainda no Capítulo 4 é apresentada a implementação de uma heurística

para o roteamento de tarefas e depois é aplicada a nova metodologia ao Sistema Modular de produção. No Capítulo 5 a metodologia desenvolvida no Capítulo 4 é aplicada a um novo módulo para validar a metodologia utilizando em conjunto com mais de um módulo. Finalmente, no Capítulo 6 são apresentadas as conclusões sobre o trabalho e as perspectivas para melhorias em trabalhos futuros.

2 CONTROLE SUPERVISÓRIO

Nesta seção, os conceitos sobre SED, TCS e sua abordagem modular local, são apresentados de forma resumida. Para uma introdução mais detalhada e didática, é indicado como bibliografia o livro de Cassandras e Lafortune (2008).

2.1 SISTEMAS A EVENTOS DISCRETOS (SED)

Um SED é um sistema dinâmico que evolui de acordo com a ocorrência repentina, possivelmente em intervalos irregulares imprevisíveis, de eventos físicos (RAMADGE; WONHAM, 1989). Um sistema é uma parte limitada do Universo que interage com o mundo externo através das fronteiras que o delimitam, os sistemas de interesse percebem as ocorrências no mundo externo através da recepção de estímulos, denominados eventos (CURY, 2001).

Um evento pode ser identificado como uma ação específica que ocorre instantaneamente e caracteriza a transição entre estados (CASSANDRAS; LAFORTUNE, 2008). A ocorrência de um evento causa uma mudança interna no sistema, que pode ser causada pela ocorrência de um evento interno ao próprio sistema, tal como o término de uma atividade ou o fim de uma temporização, onde o sistema reage imediatamente, permanecendo numa nova situação até que ocorra um novo evento.

O comportamento de um SED é modelado por um conjunto de cadeias de símbolos representando todas as possíveis sequências de eventos admitidas pelo sistema. Uma das ferramentas utilizadas para a modelagem de tais sistema é a teoria de autômatos e linguagens, a qual permite gerar a síntese automática de uma lógica de controle ótima a partir de um modelo matemático do sistema em malha aberta e de especificações de controle que atuaram na planta do sistema (QUEIROZ, 2004).

Vários modelos para SEDs foram desenvolvidos, mas nenhum é tido como modelo universal para representação, todos refletem formas distintas de representar o comportamento de uma planta. Podemos citar como principais as Redes de Petri Controladas, a Teoria das Filas, a Álgebra de Processos, a Lógica Temporal e, como já citado anteriormente, a Teoria de Linguagens e Autômatos, a qual é usada para o desenvolvimento deste projeto.

2.2 TEORIA DE CONTROLE SUPERVISÓRIO

Ramadge e Wonham (1989) propuseram a TCS, a qual é uma abordagem formal para controle de SEDs com base na teoria de autômatos e linguagens, que estabelece que os eventos que afetam o comportamento de um SED são inibidos por um agente de controle, denominado supervisor, segundo regras estabelecidas através de uma estrutura de controle.

A TCS utiliza a modelagem da planta e das especificações por linguagens de forma a permitir a síntese automática de controladores ótimos. A planta é o sistema a ser controlado e é modelada por um autômato formado por uma quíntupla $G = (\ \Sigma,\ Q,\ \delta,\ q_0,\ Q_m)$, onde Σ representa o conjunto de eventos, Q o conjunto de estados, $q_0 \in Q$ o estado inicial, $Q_m \subseteq Q$ o subconjunto de estados marcados e $\delta : \Sigma \times Q \rightarrow Q$ a função de transição parcial definida em cada estado de Q para um subconjunto de Σ.

Os autômatos podem ser representados por *diagramas de transição de estado*, como mostra a Figura 2. Os nós representam os estados possíveis em que uma máquina pode se encontrar e os arcos representam eventos responsáveis pela transição de um estado da máquina para outro. Os arcos com um pequeno corte representam eventos controláveis, os arcos sem o corte, representam eventos não controláveis. A seta representa o estado inicial da planta e a representação de um duplo circulo especifica um estado marcado.

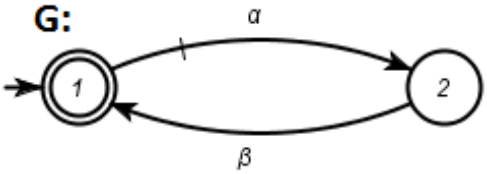

Figura 2 – Exemplo de um autômato.

A planta a ser controlada é modelada por um autômato, cujo alfabeto de eventos é particionado em dois subconjuntos: eventos controláveis (Σc) e eventos não controláveis (Σu), portanto $\Sigma = \Sigma c \bigcup \Sigma u$. Os eventos controláveis podem ser desabilitados ou habilitados em qualquer momento pelo supervisor, já os eventos não controláveis não podem ser evitados (de ocorrer) e não recebem interferência do supervisor, portanto são considerados permanentemente habilitados.

O supervisor é a entidade que controla e interage com a planta G em uma estrutura em malha fechada, onde ele observa os eventos ocorridos na planta e estabelece quais eventos controláveis são desabilitados, caracterizando uma entrada de controle que é atualizada a cada nova ocorrência de evento observada em G, de modo a coordenar seu comportamento segundo um conjunto de especificações. Um supervisor pode ser modelado por um autômato em que cada estado é associado a um conjunto de eventos que devem ser desabilitados, o supervisor é denotado por S. A Figura 3 ilustra a interação do supervisor com a planta em malha fechada.

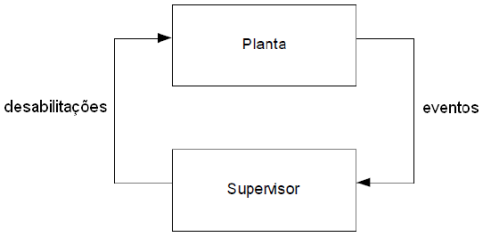

Figura 3 – Estrutura em malha fechada do controle supervisório.

O comportamento do sistema obtido pela planta G, sujeita à ação de um supervisor S, é denotada por S/G. O supervisor S é dito não-bloqueante para um autômato G, se qualquer estado de S/G acessível por uma sequência de eventos partindo do estado inicial possuir um caminho em S/G que leve a um estado marcado.

Assim como a planta G é modelada por um autômato, o conjunto de especificações também pode ser modelado por um autômato R. A TCS fornece algoritmos de complexidade polinomial no número de estados do autômato G que permitem a partir dos autômatos da planta G e da especificação R obter um supervisor não-bloqueante S, que faça com que o sistema em malha fechada S/G satisfaça as especificações. Isto é, a ação do supervisor é calculada para ser minimamente restritiva, no sentido de inibir apenas os eventos controláveis indesejados, de acordo com as especificações. Essa síntese de supervisor é chamada de cálculo de supervisor ótimo. O fato dos supervisores poderem chegar a um grande número de estados, torna a aplicação pouco prática. Uma alternativa é reduzir os supervisores representando-os por autômatos com um menor número de estados, os quais terão a mesma ação de controle sobre a planta (SU; WONHAM, 2004).

Todavia, a modelagem de plantas mais complexas com autômatos acaba sendo inviável devido ao tamanho da complexidade computacional. A seguir é apresentada a metodologia para a síntese de supervisores com base na abordagem de controle modular local, que visa superar esse problema.

2.3 CONTROLE SUPERVISÓRIO MODULAR LOCAL

Uma alternativa à abordagem da TCS em que se constrói um único supervisor monolítico para a planta é o controle modular local, proposta por Queiroz e Cury (2000), a qual permite explorar a modularidade das especificações e da planta. Isso faz diminuir a complexidade computacional da síntese de supervisores e o tamanho das soluções, decompondo o sistema físico em subsistemas modelados por autômatos que representam o seu comportamento.

Assume-se que a planta do sistema seja modelada por um conjunto de autômatos assíncronos (sem eventos em comum) $G_i, i = 1, ..., m$, e que as especificações sejam modeladas por autômatos $E_j, j, ..., n$, cada qual restringindo eventos de alguma das subplantas G_i. Segundo (CASSANDRAS; LAFORTUNE, 2008), a composição síncrona entre m autômatos é obtida com a evolução em paralelo dos m autômatos G_i, onde um evento em comum a esses m autômatos só pode ser executado de forma sincronizada nos m autômatos, enquanto que os eventos assíncronos ocorrem de modo independente em cada autômato. A composição síncrona entre dois autômatos G_1 e G_2 é denotada ao longo do documento por $G_1 \parallel G_2$.

Cada especificação E_j imposta para o sistema tem uma planta local G_{locj}, gerada pela composição síncrona dos autômatos que compartilham ao menos um evento com a especificação E_j. Em seguida é realizada a composição síncrona da planta local G_{locj} com a especificação genérica E_j que originou esta planta local, obtendo-se assim as especificações locais R_{locj}. Finalmente os supervisores locais não-bloqueantes e ótimos são calculados modularmente, i.e., um supervisor Sup_{locj} para cada especificação local R_{locj} (QUEIROZ, 2004).

Por fim, é necessário garantir a modularidade local do conjunto de supervisores locais através da verificação de que a ação conjunta de todos os supervisores seja não-bloqueante. O teste de modularidade local é feito através da composição síncrona de todos os supervisores modulares locais. Caso o autômato resultante dessa composição seja não-bloqueante, garante-se que a ação conjunta dos supervisores mo-

dulares locais é ótima, i.e., equivalente ao supervisor monolítico não-bloqueante e minimamente restritivo. Caso haja conflito entre os supervisores, deve-se adotar uma estratégia para resolução de conflito, como a síntese de um supervisor adicional (coordenador) com o único objetivo de desabilitar os caminhos que causem bloqueio (QUEIROZ; CURY, 2002a; QUEIROZ, 2004).

A Figura 4 ilustra como são definidos os supervisores locais. Os autômatos G_0, G_1, G_2 e G_3 representam os subsistemas da planta e E_0, E_1 e E_2 as especificações genéricas de um determinado sistema. Cada especificação está associada a um conjunto G_{locj} formado pelos índices dos subsistemas que têm ao menos um evento em comum com esta especificação. Assim, E_0 está associada ao conjunto $G_{locE0} = \{G_0, G_1\}$, já E_1 está associada ao conjunto $G_{locE1} = \{G_1, G_2\}$, e a E_3 está associada ao conjunto $G_{locE2} = \{G_2, G_3\}$.

Para obter as especificações locais G_{locj} referentes a cada especificação, basta realizar a composição síncrona dos elementos de cada conjunto. Cada especificação local R_{locj} é obtida pela composição síncrona de E_j com G_{locj}. A partir de R_{locj} e G_{locj}, calculam-se os supervisores modulares locais S_{locj}, $j = 1, ..3$, que são minimamente restritivos e não-bloqueantes (localmente ótimos). Por fim, deve-se testar se a composição síncrona dos supervisores é não-bloqueante para garantir que a ação de controle modular local seja ótima. Cada supervisor pode ser reduzido pelo algoritmo de Su e Wonham (2004).

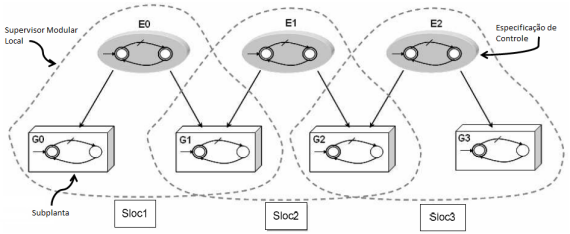

Figura 4 – Exemplo de obtenção de supervisores locais.

2.4 ARQUITETURA DE CONTROLE SUPERVISÓRIO

Para organizar a implementação da abordagem modular local é definida uma estrutura com três níveis hierárquicos (QUEIROZ; CURY, 2002b), de acordo com a Figura 5. A implementação desta estru-

tura pode ocorrer em linguagem de CLP (IEC 61131-3, 1998), de PC (C, Java) ou mesmo diretamente em *hardware* (circuito elétrico, pneumático ou hidráulico).

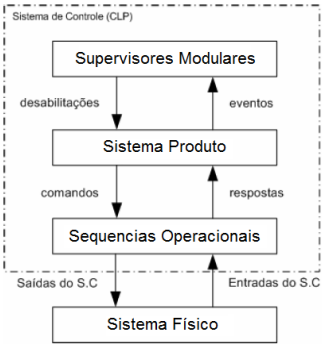

Figura 5 – Arquitetura de Controle Supervisório.
Fonte: (QUEIROZ; CURY, 2002b).

No nível mais alto (Supervisores Modulares) implementa-se o conjunto de supervisores modulares locais reduzidos na forma de autômatos, e que são atualizados a cada ocorrência de um evento gerado na planta. Um mapa de realimentação associa os estados ativos a um conjunto de sinais de desabilitações que controlam o Sistema Produto (representação do subsistema da planta).

No nível do Sistema Produto as principais funções são disparar os comandos da planta que não foram desabilitados pelos Supervisores e receber respostas enviadas pelas sequências operacionais. A evolução (mudança de estado) dos supervisores ocorre juntamente com a do Sistema Produto através de seus comandos (eventos controláveis) e respostas da planta (eventos não controláveis), abstraídas nas Sequências Operacionais.

As sequências operacionais são responsáveis por interpretar os comandos do Sistema Produto e gerar as instruções para os atuadores da planta, e interpretar os sinais dos sensores da planta, sintetizando e sinalizando a ocorrência de eventos não controláveis. O nível de Sequências Operacionais funciona como uma interface entre o Sistema Produto e o Sistema Físico.

3 APLICAÇÃO DA METODOLOGIA PARA DESENVOLVIMENTO INTEGRADO DE SISTEMAS SCADA COM CONTROLE SUPERVISÓRIO

Os sistemas SCADA utilizam tecnologias de computação e comunicação para automatizar o monitoramento e o controle de processos industriais. Estes sistemas são parte integrante da maioria dos ambientes industriais complexos ou geograficamente dispersos, na medida em que podem coletar rapidamente os dados de uma grande quantidade de fontes, para depois serem apresentados a um operador de uma forma amigável (PORTILLA, 2011).

A principal função de um sistema SCADA é propiciar uma interface de alto nível do operador com o processo, informando-o em tempo real, através de um conjunto de telas, gráficos e relatórios sobre todos os eventos importantes ocorridos no sistema controlado, permitindo a tomada de decisões operacionais apropriadas (PINHEIRO, 2006).

Portilla (2011) apresenta uma metodologia para integrar o sistema SCADA com o controle supervisório, utilizando a AMCS implementada em dispositivos de controle, como o CLP. A metodologia de Portilla (2011) foi desenvolvida com base no trabalho de Silva e Queiroz (2009), que estabelecem as seguintes etapas para o controle de um FMS: modelagem da planta e especificações; síntese de supervisores; emulação de geradores e implementação segundo a ACS.

3.1 METODOLOGIA PARA DESENVOLVIMENTO INTEGRADO DE SISTEMAS SCADA COM CONTROLE SUPERVISÓRIO

O trabalho de Portilla (2011) explora a integração de funcionalidades do sistema SCADA com o controle supervisório, resultando em uma metodologia de desenvolvimento integrado dividida em oito fases: projeto informacional; síntese de controle supervisório; emulação; implementação de controle supervisório em CLP; implementação de funcionalidades básicas do SCADA; avaliação de funcionamento do sistema real; implementação de funcionalidades gerais do sistema SCADA; e validação. A Figura 6 apresenta esta metodologia.

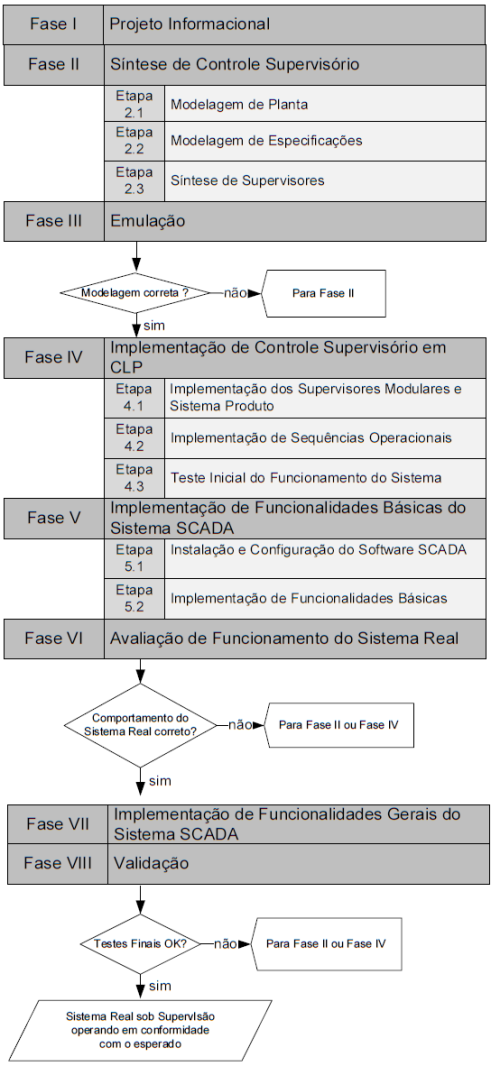

Figura 6 – Metodologia para integração de Controle Supervisório a um Sistema SCADA.

Fonte: (PORTILLA, 2011).

3.1.1 Projeto Informacional

A primeira fase da metodologia está relacionada com o levantamento das especificações técnicas do projeto, com o intuito de facilitar

o desenvolvimento da modelagem da planta.

É necessário identificar os subsistemas e o problema de controle da planta para definir as restrições do comportamento e coordenação desejados para o sistema. Dispositivos de campo como sensores e atuadores, tecnologias a serem utilizadas no desenvolvimento do projeto para controle da planta e protocolo de comunicação a ser usado para estabelecer conexão entre o equipamento de controle e a estação de supervisão devem constar no levantamento do projeto informacional. Por fim define-se as ferramentas utilizadas para a síntese do controle supervisório segundo a TCS (PORTILLA, 2011).

3.1.2 Síntese de controle supervisório

De acordo com o TCS, a Síntese de Controle Supervisório é composta por três etapas: obtenção de um modelo na forma de autômatos representando o funcionamento da planta; modelagem das especificações genéricas de controle a fim de expressar matematicamente o comportamento desejado do sistema; e síntese dos supervisores não-bloqueantes (CASSANDRAS; LAFORTUNE, 2008; WONHAM, 2008).

Na modelagem da planta são construídos os autômatos que representam o comportamento independente de cada subsistema. Deve-se considerar a maior abstração possível a fim de diminuir o número de estados e o custo computacional. Os sistemas de maior complexidade são modelados por diversos subsistemas concorrentes interagindo entre si.

A planta por si só pode executar qualquer ação em malha aberta, devido à interação sem controle dos diversos subsistemas. Com o intuito de restringir o comportamento da planta evitando estados proibidos, são definidas especificações, as quais impossibilitam a planta de executar algumas ações. Para cada especificação estabelecida, é gerado um autômato.

Por último realiza-se a síntese de supervisores locais. Para isso são realizadas as seguintes etapas: é obtida primeiramente uma planta local para cada especificação; depois realiza-se o produto síncrono da planta local com a sua respectiva especificação; em seguida calcula-se o supervisor ótimo contido em cada especificação local; depois realiza-se a redução dos supervisores locais; e por fim faz-se o teste de não conflito. O teste de não conflito assegura que a ação conjunta dos supervisores modulares é globalmente não-bloqueante e ótimo. Esta etapa segue a abordagem modular local desenvolvida por Queiroz e Cury (2000).

3.1.3 Emulação

A Emulação é a fase para realizar os primeiros testes da lógica de controle. Caso a modelagem não esteja de acordo com a lógica proposta ou erros sejam encontrados, é possível voltar para a fase dois para realizar as mudanças necessárias. Quando a modelagem se encontrar em conformidade com a lógica proposta, os supervisores e o sistema produto estão prontos para serem implementados no controlador.

3.1.4 Implementação do controle supervisório em CLP

A fase de implementação do controle supervisório em CLP é realizada segundo o método de implementação desenvolvido por Queiroz e Cury (2002b), e é dividida em três etapas. A primeira refere-se a implementação da lógica de controle supervisório. A segunda etapa é a implementação das sequências operacionais definida pelo programador no CLP. A última etapa são os testes iniciais do funcionamento do sistema.

A implementação da lógica de controle supervisório, modelado na segunda fase da metodologia, é dividida em duas partes: os Supervisores Modulares reduzidos, responsáveis pelo controle da evolução do sistema; e os subsistemas que integram a planta, denominados Sistemas Produtos.

Em seguida, implementam-se as sequências operacionais, com as funções de ler os sinais que ocorrem na planta, como término de uma ação ou chegada de uma peça, e gerar os comandos dos atuadores da planta. Esses sinais são lidos e escritos nas entradas e saídas do CLP. A linguagem de implementação dessa etapa fica a critério do programador, desde que seja obedecida a norma International... (1998).

Na última etapa é realizada uma verificação inicial para encontrar erros na lógica. Como nesta etapa ainda não foram inseridos os dispositivos de campo, a verificação pode não ser exaustiva, ficando para fases posteriores uma verificação mais aprofundada.

3.1.5 Implementação de funcionalidades básicas do sistema SCADA

As funcionalidades como sinótico, envio de comandos, histórico de eventos e a geração de alarmes críticos, são denominadas funcionali-

dades básicas de um sistema SCADA. Elas devem fornecer informações gráficas e textuais da planta que está sendo controlada. Possíveis erros de modelagem que não foram identificados nas fases anteriores, podem ser reconhecidos (PORTILLA, 2011).

O primeiro passo da quinta fase da metodologia é separado para a instalação e configuração do *software* SCADA. Após a instalação, é configurado o protocolo de comunicação entre o CLP e o SCADA. Para verificar se a comunicação entre os *softwares* está ocorrendo, é realizado um teste de troca de dados.

No segundo passo são implementadas todas as funcionalidades básicas do sistema SCADA, como sinótico, envio de comandos, histórico de eventos e alarmes críticos. Essas funcionalidades são implementadas com base no controle supervisório implementado no CLP.

3.1.6 Avaliação de funcionalidades do sistema real

Nesta fase são realizados os testes de comportamento real da planta, já com os atuadores e sensores instalados. Os resultados devem estar em conformidade com o planejamento inicial. Caso alguma incoerência seja encontrada, deve-se voltar a fase dois ou quatro, conforme a necessidade, para realizar as correções. Assim que os testes desta fase encontrarem-se satisfatórios, a fase de avaliação de funcionalidades do sistema real é concluída (PORTILLA, 2011).

3.1.7 Implementação de funcionalidades gerais do sistema SCADA

As funcionalidades gerais utilizadas em um sistema SCADA, são aquelas que ampliam e complementam as informações sobre o processo da planta, outros setores da empresa podem utilizar essas informações para aperfeiçoar e controlar a produção, normalmente em forma de gráficos. Informações como erros e alarmes também devem ser implementados, com o intuito de aumentar a segurança da planta e dos operadores. As principais funcionalidades que podem ser implementadas nessa fase são: geração de alarmes gerais, gráficos de tendências, receitas, relatórios e geração de informações para níveis gerenciais. A escolha de quais funções devem ser implementadas é feita tendo em vista as necessidades da empresa.

3.1.8 Validação

Após a implementação de todas as funcionalidades é realizado o teste final, a fim de validar o funcionamento do sistema de manufatura, sendo controlado pelo CLP e monitorado pelo SCADA. Caso os testes não se encontrem em conformidade com as especificações, deve-se analisar o problema e voltar para a segunda ou quarta fase, conforme for a necessidade. Quando os resultados dos testes forem positivos, a implementação do sistema SCADA integrado com controle supervisório está completa e validada. Esta é a última fase da metodologia proposta por Portilla (2011).

3.2 ESTAÇÃO DE DISTRIBUIÇÃO

Neste capítulo é aplicada a metodologia desenvolvida por Portilla (2011) em uma planta MPS da Festo, localizada no Laboratório de Automação Industrial (LAI) da UFSC. Dentre os módulos que compõem a planta MPS, está o módulo Distribuição, o qual é usado para aplicar a metodologia explicada na seção 3.1. O Sistema de Aprendizagem Festo Didática de Automação e Tecnologia foi projetado para atender uma série diferente de treinamentos e exigências profissionais. Os módulos do MPS facilitam a formação orientada para a indústria e o *hardware* é composto de componentes industriais didáticos (EBEL; PANY, 2006).

Cada módulo do MPS é controlado individualmente por um CLP da marca Siemens da série S7-1200 com 14 entradas digitais e 10 saídas digitais, existe ainda um módulo expansão que disponibiliza mais 8 entradas digitais e 8 saídas digitais.

O módulo Distribuição trabalha com o objetivo de coletar peças que chegam e encaminha-las para um outro módulo. A cada chegada de uma nova peça, um cilindro pneumático (alimentador) tem a função de empurra a peça para a posição P1, como é visto na Figura 7. Existe um braço giratório que contém na ponta uma ventosa para prender a peça, o braço rotaciona e leva a peça para a posição P2, posição esta que fica no módulo seguinte do MPS (Estação de Teste). Na posição P2, a peça será encaminhada pelo módulo Estação de Teste do MPS para ser analisada. Existe um sensor na forma de barreira luminosa, chamado de IP_FI, que está configurado para receber do módulo seguinte um sinal para bloquear o avanço do braço, isto ocorre quando o módulo seguinte está executando alguma tarefa.

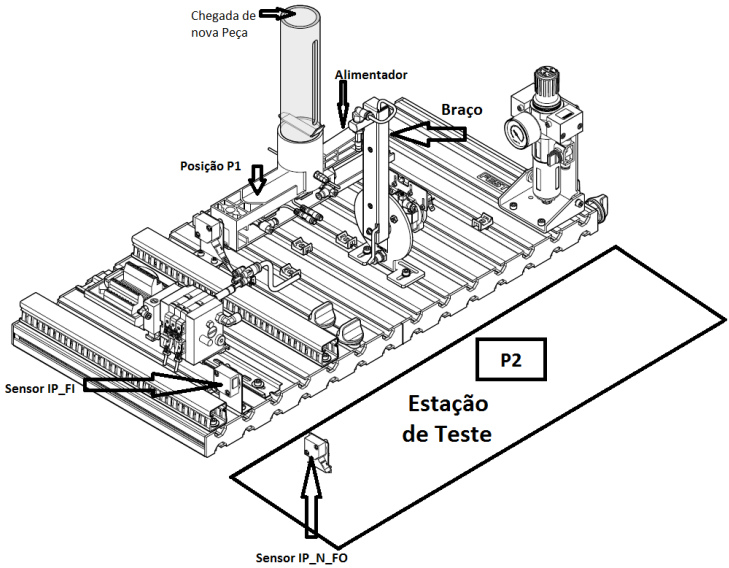

Figura 7 – MPS Festo - módulo 1.
Fonte: (EBEL; PANY, 2006).

3.3 APLICAÇÃO DA METODOLOGIA

Para o módulo Distribuição, propõe-se modelar uma lógica de controle utilizando a TCS, implementar essa lógica no CLP e desenvolver um sistema SCADA. Esse desenvolvimento segue a metodologia desenvolvida por (PORTILLA, 2011) e todas as oito fases são implementadas e explicadas a seguir.

3.3.1 Projeto Informacional

Antes de começar a modelar o sistema, é necessário realizar um levantamento das informações técnicas referente à planta. O módulo Distribuição é composto por:

- 1 sensor que identifica a chegada de uma nova peça;
- 1 cilindro pneumático com retorno por mola que leva as peças para a posição P1;

- 1 sensor indutivo anexo ao cilindro pneumático que identifica quando o alimentador está avançado ou quando está recuado;
- 1 braço giratório que transporta a peça da posição P1 para a posição P2;
- 2 sensores fim-de-curso que identificam quando o braço avançou e recuou;
- 1 ventosa para segurar a peça;
- 1 Sensor de barreira luminosa (IP_FI) que está relacionado com o bloqueio/liberação do módulo seguinte;
- 1 CLP Siemens S7-1200;
- 1 painel de IHM;

As ferramentas utilizadas na aplicação do controle supervisório são: o *software* SUPREMICA, desenvolvido por AKESSON (2002) para realizar a construção de autômatos, tanto da planta quanto das especificações, e para emulação do controle supervisório; o IDES (NGUYEN et al.,) e o TCT (FENG; WONHAM, 2006) para realizar a redução dos supervisores; O IDES2ST (KLINGE, 2007) para gerar de forma automática o código em linguagem Texto Estruturado (*Structured Text* - ST) (International... (1998)); Para programação do CLP e implementação do *SCADA*, são utilizados os *softwares* TIA Portal (*Totally Integrated Automation Portal*) e WinCC (SIEMENS, 2012) respectivamente. A comunicação entre SCADA e CLP é realizada via Ethernet Industrial.

3.3.2 Síntese de Controle Supervisório

A metodologia proposta na Seção 3.1 segue a abordagem modular local (QUEIROZ; CURY, 2002b), que explora a modularidade da planta e das especificações dos sistemas, a fim de evitar a complexidade computacional, comum na síntese de um supervisor. A abordagem é dividida em três partes: (*i*) Modelagem dos subsistemas do MPS; (*ii*) Modelagem das especificações; (*iii*) Síntese de supervisores.

(i) - Modelagem dos subsistemas do MPS.

Cada módulo do MPS é composto por diversos subsistemas concorrentes que interagem entre si para a realização das tarefas. Cada subsistema é representado como um Sistema Produto e são assíncronos

entre sí. O módulo Distribuição é dividido em cinco subsistemas: braço giratório; alimentador de peças; sistema de sucção de peças; controle do módulo seguinte; sistema que controla a chegada de novas peças. Cada subsistema é modelado na forma de um autômato.

Em toda a planta podem ocorrer 15 eventos, sendo 6 controláveis e os demais não controláveis. Na Tabela 1 são descritos os eventos utilizados na modelagem dos subsistemas e o significado de cada um.

Tabela 1 – Eventos utilizados

Evento	**Descrição**
aliment_avanca	cilindro pneumático começa avançar
aliment_avancou	cilindro pneumático avançou
aliment_retorna	cilindro pneumático começa retornar
aliment_retornou	cilindro pneumático retornou
braco_avanca	braço inicia avanço
braco_avancou	braço avançou
braco_recua	braço inicia recuo
braco_recuou	braço recuou
chegou_P	indica chegada de uma nova peça
liberou_2	indica fim de operação do módulo seguinte
ocupou_2	indica inicio de operação do módulo seguinte
vent_pega	gerador de vácuo é acionado
vent_pegou	indica pressão atingida, i.e., peça presa
vent_ejeta	ventosa aciona soprador para ejetar a peça
vent_ejetou	ventosa ejetou a peça

Na modelagem dos subsistemas e das especificações, os círculos representam os estados, aqueles com duplos círculos identificam os estados marcados e os arcos representam a transição de um estado para outro devido a ocorrência de um evento, sendo os arcos com um traço eventos controláveis e os arcos sem traço eventos não controláveis. As modelagens de todos os subsistemas são apresentadas sa sequência.

1. Sistema que controla a chegada de novas peças.

   Descrição: Um sensor identifica a chegada de uma nova peça na posição Pcp (posição de chegada de peça) enviando um sinal ao sistema. Esse sinal representa a ocorrência do evento não controlável *Chegou_P*, ver Figura 8.

2. Alimentador de peças.

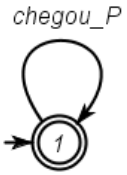

Figura 8 – Sistema Produto 1 - $SP_1\_NovaPeca$.

Descrição: Quando chega uma nova peça, um cilindro pneumático deve empurrar a mesma para o *buffer* da posição P1, esse *buffer* tem capacidade de 1 peça. Dois sensores eletromagnéticos acoplados ao pistão indicam o posicionamento do atuador (avançado ou recuado). A mudança de posição do cilindro ocorre por meio do evento *aliment_avanca*, o que faz com que o cilindro avance, ou o evento *aliment_retorna*, fazendo com que o cilindro recue, ambos eventos são controláveis. Para indicar que o cilindro avançou ou que o cilindro recuou são utilizados os eventos não controláveis *aliment_avancou* e *aliment_recuou*, como pode ser analisado na Figura 9.

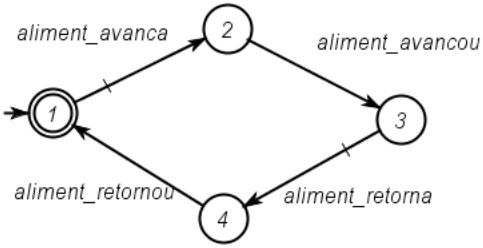

Figura 9 – Sistema Produto 2 - $SP_2\_Alimentador$.

3. Braço giratório.

   Descrição: O evento *braco_avanca* é o comando que faz o braço giratório avançar da posição P1 para a posição P2 e *braco_recua* o comando de recuar o braço da posição P2 para a posição P1. Existem dois sensores fim-de-curso, um em cada lado do braço, que indicam o momento em que o braço chega na posição desejada (avançado ou recuado), os sensores enviam sinais para o sistema, representados pelos eventos *braco_avancou* e *braco_recuou*, como pode ser visto na Figura 10.

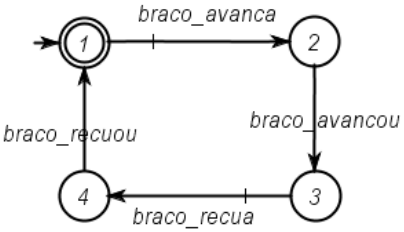

Figura 10 – Sistema Produto 3 - SP_3_Braco.

4. Sistema de sucção de peças.

   Descrição: Na ponta do braço giratório, existe um efetuador na forma de ventosa, que através de um gerador de vácuo, suga a peça para que o braço possa transportá-la da posição P1 para a posição P2. Na posição P1, o evento *vent_pega* dá o comando para acionar o gerador de vácuo e o evento *vent_pegou* indica que a peça foi pega. O braço transporta a peça para a posição P2, onde ela deve ser ejetada, por meio do comando *vent_ejeta*. O evento *vent_ejetou* indica que a peça foi solta, como é ilustrado na Figura 11.

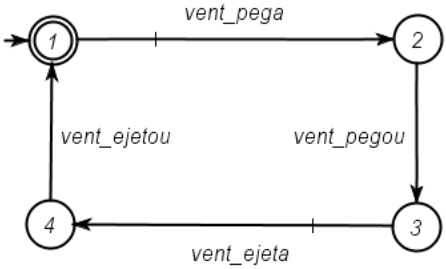

Figura 11 – Sistema Produto 4 - $SP_4\_Ventosa$.

5. Controle do módulo seguinte.

   Descrição: Quando o módulo que vêm na sequência está executando alguma ação, ele deve mandar um sinal para o módulo Distribuição indicando estar ocupado. Esse sinal é recebido pelo sensor de barreira luminosa, o qual repassa a informação para o sistema. O evento *ocupou_2* indica o momento em que o módulo seguinte está executando alguma ação e o evento *liberou_2* indica

que o módulo seguinte terminou de executar as ações. Ambos eventos são não controláveis por dependerem do módulo 2, como é visto na Figura 12.

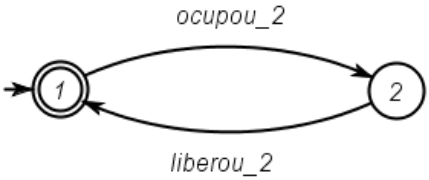

Figura 12 – Sistema Produto 5 - $SP_5\_ModuloSeguinte$.

(ii) - Modelagem das especificações.

Para o funcionamento adequado do sistema de manufatura, algumas especificações são definidas para restringir o comportamento da planta e evitar situações de bloqueio. Elas são:

- O alimentador não avança enquanto o braço não estiver avançado;
- A ventosa não ativa se o braço não estiver recuado (posição P1) e não ejeta se o braço não estiver avançado (posição P2);
- Caso a ventosa esteja segurando uma peça, o recuo do braço fica bloqueado até a ventosa ejetar essa peça;
- A ventosa não ativa enquanto o alimentador não colocar uma peça na posição P1;
- Exclusão mútua entre o recuo do braço e o avanço do alimentador;
- Exclusão mútua entre o avanço do braço e a ativação da ventosa;
- A ventosa não ativa se o alimentador não estiver recuado;
- O braço não recua enquanto o alimentador não colocar uma peça na posição P1 ou a ventosa soltar uma peça na posição P2 (essa especificação evita que o braço recue desnecessariamente);
- O alimentador não avança enquanto não chegar uma nova peça;
- O braço não pode avançar caso o módulo seguinte esteja em operação (informação indicada pelo sensor de barreira luminosa).

A seguir é apresentada a modelagem de todas as especificações.

1. O alimentador não avança enquanto o braço não estiver avançado.

   Descrição: Para garantir que não haja conflito entre o braço e o alimentador foi definido, como visualizado na Figura 13, que o alimentador não deve avançar enquanto o braço estiver na posição P1, ou seja, no estado 1 da Figura 13 o alimentador fica impossibilitado de avançar até chegar o sinal de que o braço avançou para a posição P2.

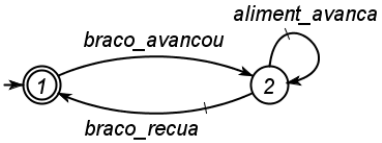

Figura 13 – Especificação 1 - $E_1\_AvancoAlimentador$.

2. A ventosa não ativa se o braço não estiver retraído (posição P1) e não ejeta se o braço não estiver avançado (posição P2).

   Descrição: A ventosa executa as ações de pegar e ejetar peças. O comando de pegar peça não deve ocorrer na posição P2 e o de ejetar peça não deve ocorrer na posição P1. Na Figura 14 é garantido que no estado 2 o evento *vent_pega* não ocorre, da mesma forma que no estado 1 o evento *vent_ejeta* também não ocorre.

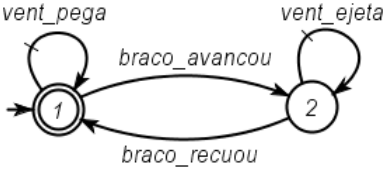

Figura 14 – Especificação 2 - $E_2\_ControleVentosa$.

3. Caso a ventosa esteja com uma peça, o recuo do braço fica bloqueado até a ventosa ejetar a peça.

   Descrição: Após a ventosa ter pegado uma peça e o braço avançar para a posição P2, o mesmo não deve retornar até que a ventosa

ejete a peça, como mostrado na Figura 15, ou seja, estando no estado 2, o braço só pode voltar para a posição P1 após a ocorrência do evento *vent_ejetou*.

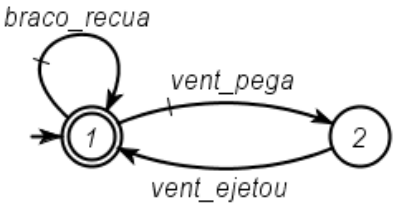

Figura 15 – Especificação 3 - $E_3\_EjetarParaRecuar$.

4. A ventosa não ativa enquanto o alimentador não colocar uma peça na posição P1.

   Descrição: O braço retorna para o posição P1 em diversos momentos, em alguns desses a ventosa deve acionar para pegar uma peça. A lógica deve controlar o momento em que a ventosa aciona, a fim de evitar que ela ative desnecessariamente ou que ela deixe de ativar quando houver a necessidade. Assim, a ventosa não deve acionar se não houver peça na posição P1, sendo necessária a ocorrência do evento *aliment_avancou*, e o braço não deve avançar sem entes a ventosa pegar a peça, como ilustrado na Figura 16. Enquanto estiver no estado 1, não é permitida a ocorrência do evento *vent_pega*.

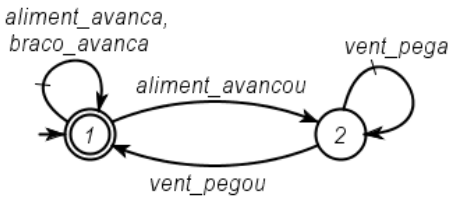

Figura 16 – Especificação 4 - $E_4\_ControleAcionaVentosa$.

5. Exclusão mútua entre o recuo do braço e o avanço do alimentador.

   Descrição: Para garantir que a ação de recuar o braço não ocorra junto com a ação de avançar o alimentador, é definida uma exclusão mútua entre os dois eventos. Na Figura 17 pode-se ver que no estado 2 pode ocorrer tanto o comando para recuar o

braço quanto o de avançar o alimentador. Caso ocorra o evento *braco_recua*, fazendo a transição do estado 2 para o estado 1, o alimentador só pode avançar depois que ocorrer o evento *braco_recuou*, o que resulta em retornar para o estado 2. O mesmo ocorre se o evento *aliment_avanca* executar, o recuo do braço fica bloqueado até o evento *aliment_avancou* ocorrer.

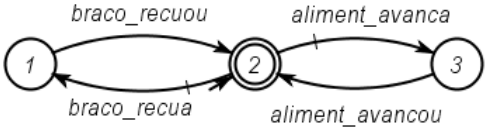

Figura 17 – Especificação 5 - $E_5\_Braco\_OU\_Alimentador$.

6. Exclusão mútua entre o avanço do braço e a ativação da ventosa.

   Descrição: Para não haver o problema de o braço avançar sem que a peça esteja completamente presa, indicada pelo evento *vent_pegou*, e que a ventosa não acione enquanto o braço estiver em transição, é especificada uma exclusão mútua entre o braço e a ventosa. Na Figura 18 pode ser visto que no estado 2 pode ocorrer tanto o comando para acionar o braço, quanto o de acionar a ventosa. Caso ocorra o evento *braco_avanca*, ocasionando na mudança do estado 2 paro o 1, a ventosa só pode ser acionada se ocorrer o evento *braco_avancou*, o que resulta em retornar para o estado 2 e liberar a ventosa para acionar. O mesmo ocorre se o evento *vent_pega* executar, o avanço do braço fica bloqueado até o evento *vent_pegou* ocorrer.

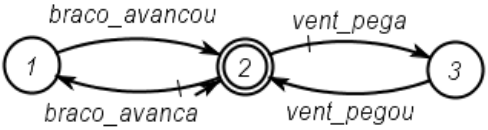

Figura 18 – Especificação 6 - $E_6\_Braco\_OU\_Ventosa$.

7. A ventosa não ativa se o alimentador não estiver recuado.

   Descrição: Quando o alimentador avança, ele carrega uma peça e a prende contra uma parede. Se nesse momento o braço voltar para a posição P1, a ventosa irá acionar para pegar a peça e

quando a peça estiver pega o sensor indicará que a ventosa pegou a peça, então o braço estará liberado para avançar e levar a peça para a posição P2. Se isso ocorrer, a ventosa não conseguirá retirar a peça devido ao fato de o alimentador ainda estar pressionando a peça, então ocorrerá um erro na lógica. Para resolver esse problema é modelada a especificação 7, onde o estado 1 libera o acionamento do evento *vent_pega*, se ocorrer o evento *aliment_avancou*, o que implica em mudar para o estado 2, o acionamento da ventosa estará bloqueado até que o alimentador retorne. Essa lógica evita que a ventosa acione enquanto o alimentador não estiver recuado. Essa especificação é ilustrada na Figura 19.

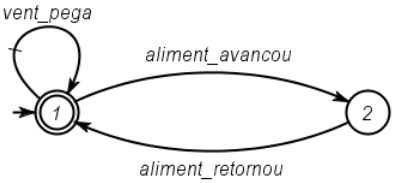

Figura 19 – Especificação 7 - $E_7\_EvitarPecaPresa$.

8. O braço não recua enquanto o alimentador não colocar uma peça na posição P1 ou a ventosa soltar uma peça na posição P2.

   Descrição: Em alguns momentos não haverá peça tanto na posição P1 quanto na posição P2, nesse momento o braço poderá entrar em um *Loop* e ficar rotacionando de um lado para o outro. A especificação 8 evita esse *Loop*, apenas restringindo a ação de retornar o braço, a menos que ocorra um entre dois eventos que fazem necessária a utilização do recuo do braço. Os dois eventos são: *vent_ejetou* e *aliment_avancou*. O primeiro refere-se ao momento em que a ventosa ejetou uma peça na posição, sendo necessário que o braço recue para liberar o trabalho do módulo seguinte. Já o segundo refere-se ao momento em que o alimentador avançou, o que indica a presença de uma peça na posição P1 e a necessidade de retornar o braço para buscar essa peça. Enquanto não ocorrer pelo menos um desses dois eventos, o evento *braco_recua* fica impossibilitado de ocorrer, bloqueando a ação de recuar o braço, como é ilustrado na Figura 20.

9. O alimentador não avança enquanto não chegar uma nova peça.

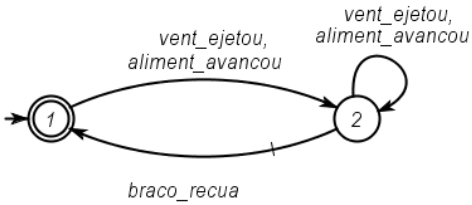

Figura 20 – Especificação 8 - E_8_$RecuoBraco$.

Descrição: O alimentador deve avançar somente quando for para levar uma peça para a posição P1. A especificação 9, visualizada na Figura 21, define que o alimentador não deve avançar sem que ocorra o evento *chegou_P* indicando a chegada de uma nova peça, ou seja, no estado 1 o comando *aliment_avanca* é impossibilitado de ocorrer.

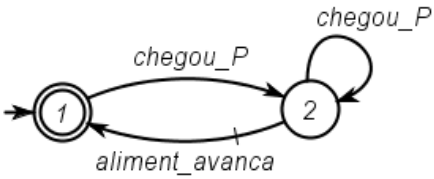

Figura 21 – Especificação 9 - E_9_$ControleChegadaPeca$.

10. O braço não pode avançar caso o módulo seguinte esteja em operação.

    Descrição: O MPS da Festo tem uma configuração através da qual o módulo sempre leva em conta o estado do módulo que está à sua frente. Isto permite a execução do trabalho de um módulo somente se o módulo da frente estiver parado. No módulo Distribuição, quando a posição P2 fica ocupada, o braço deve voltar para a posição P1 para liberar a operação do módulo seguinte. No momento em que o módulo seguinte começa a executar, ele manda um sinal de ocupado e o evento *braco_avanca* deve ser bloqueado pela lógica de controle. Na Figura 22 é possível ver que quando ocorre o evento *ocupou_2*, fazendo a transição do estado 1 para o estado 2, o avanço do braço é bloqueado. Quando o módulo seguinte termina o trabalho, o evento *liberou_2* é executado, o que resulta em retornar para o estado 1 e liberar o avanço

do braço.

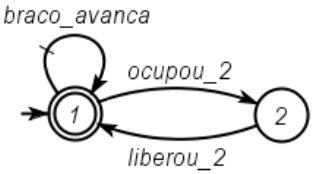

Figura 22 – Especificação 10 - $E_{10}\_ModuloSeguinte$.

(iii) - Síntese de Supervisores.

Para a síntese dos supervisores é utilizada a abordagem modular local. Primeiramente são obtidas as plantas locais G_{locj} por meio da composição dos autômatos dos subsistemas G_i (também denotados por SP_i) afetados pelas especificações genéricas E_j, ou seja, faz-se a composição daqueles que compartilham ao menos um evento com a especificação. As plantas locais para o MPS são vistas na Tabela 2.

Tabela 2 – Plantas locais

G_{locj}	$SP_1 \| SP_2 \| ... \| SP_i$
G_1	$SP_2\_Alimentador \parallel SP_3\_Braco$
G_2	$SP_3\_Braco \parallel SP_4\_Ventosa$
G_3	$SP_3\_Braco \parallel SP_4\_Ventosa$
G_4	$SP_2\_Alimentador \parallel SP_3\_Braco \parallel SP_4\_Ventosa$
G_5	$SP_3\_Braco \parallel SP_2\_Alimentador$
G_6	$SP_3\_Braco \parallel SP_4\_Ventosa$
G_7	$SP_4\_Ventosa \parallel SP_2\_Alimentador$
G_8	$SP_2\_Alimentador \parallel SP_3\_Braco \parallel SP_4\_Ventosa$
G_9	$SP_1\_NovaPeca \parallel SP_2\_Alimentador$
G_{10}	$SP_3\_Braco \parallel SP_5\_ModuloSeguinte$

Como exemplo, tomamos a especificação mostrada na Figura 13, a qual utiliza os seguintes eventos na modelagem: *aliment_avanca*; *braco_avancou* e *braco_recua*. Por conta da utilização desses eventos, a especificação afeta os seguintes subsistemas: $SP_2\_Alimentador$ e SP_3_Braco. Em cima destes dados, é realizada a composição dos subsistemas assíncronos afetados pela especificação, gerando a planta local: $(G_1 = SP_2\_Alimentador \parallel Sp_3\_Braco)$.

O segundo passo é calcular a especificação local R_{locj} para cada planta local G_{locj}. Esse passo é realizado fazendo-se a composição

síncrona de cada planta local com a especificação em comum: ($R_{locj} = G_{locj} \| E_j$). A Tabela 3 descreve todas as composições realizas.

Tabela 3 – Especificações locais

R_{locj}	$G_{locj} \parallel E_j$
R_1	$G_1 \parallel E_1\_AvancoAlimentador$
R_2	$G_2 \parallel E_2\_ControleVentosa$
R_3	$G_3 \parallel E_3\_EjetarParaRecuar$
R_4	$G_4 \parallel E_4\_ControleAcionaVentosa$
R_5	$G_5 \parallel E_5\_Braco\_OU\_Alimentador$
R_6	$G_6 \parallel E_6\_Braco\_OU\_Ventosa$
R_7	$G_7 \parallel E_7\_EvitarPecaPresa$
R_8	$G_8 \parallel E_8\_RecuoBraco$
R_9	$G_9 \parallel E_9\_ControleChegadaPeca$
R_{10}	$G_{10} \parallel E_{10}\_ModuloSeguinte$

Seguindo com o exemplo, após gerar a planta local G_1, é gerada a especificação local R_1, através da composição da planta local obtida anteriormente com a respectiva especificação ilustrada na Figura 13: $R_1 = G_1 \parallel E_1\_AvancoAlimentador$.

O terceiro passo é calcular o supervisor ótimo para cada especificação local R_x e planta local G_x sendo $x = 1, ..., 10$. Esta etapa gera um supervisor local Sup_j para cada planta local. Foram gerados 10 supervisores locais para o módulo Distribuição do MPS, sendo todos não-bloqueantes. Nos exemplos utilizados anteriormente, foram encontradas a planta local G_1 e a especificação local R_1, agora é necessário encontrar o supervisor ótimo. Então faz-se a síntese de R_1 com G_1, gerando o supervisor local: $Sup_1 = (R_1, G_1)$.

Para garantir o não conflito da ação de todos os supervisores locais em conjunto é verificado o comportamento em conjunto e concorrente dos supervisores, garantindo que a ação de todos os supervisores seja não-bloqueante. Assim, é calculado o autômato S por meio da sincronização de todos os supervisores locais ($S = Sup_1 \parallel Sup_2 \parallel ... \parallel Sup_j$). A sincronização dos 10 supervisores encontrados no terceiro passo, retornou um autômato S não-bloqueante com 232 estados e 776 transições.

Todas as operações envolvendo a síntese dos supervisores, foram realizadas primeiramente na ferramenta Supremica devido à possibilidade de pequenos testes e simulações desta síntese. Depois foi realizada a modelagem com a ferramenta IDES, visando a geração automática do código dos supervisores em ST para implementação em CLP.

Mesmo utilizando a abordagem modular local, alguns supervisores possuem um grande número de estados devido a explosão do número de estados na síntese dos supervisores modulares locais. Isso torna a aplicação pouco prática, visto que necessita de uma grande quantidade de memória computacional. Caso o valor de R_{locj} seja igual ao Sup_j, pode-se usar diretamente a especificação genérica E_j como um supervisor reduzido, pois já estará próxima do supervisor reduzido ótimo e os mesmos terão a mesma ação de controle sobre a planta, mas com um número consideravelmente menor de estados. Como todas as especificações locais R_{locj} do módulo Distribuição são ótimas ($Sup_j = R_{locj}$), os autômatos das respectivas especificações genéricas podem ser diretamente adotados como supervisores reduzidos sem a necessidade de usar o algoritmo de redução do TCT.

O número de estados de todos os autômatos envolvidos na síntese de supervisores modulares locais, bem como os supervisores locais reduzidos, estão descritos na Tabela 4.

Tabela 4 – Número de estados dos autômatos da síntese

j	E_j	G_{locj}	R_{locj}	Sup_j	$RSup_j$
1	2	16	16	16	2
2	2	16	16	16	2
3	2	16	16	16	2
4	2	64	96	96	2
5	3	16	15	15	3
6	3	16	15	15	3
7	2	16	16	16	2
8	2	64	128	128	2
9	2	8	16	16	2
10	2	8	8	8	2

Onde:

- R_{locj} = Especificação local;
- E_j = Especificação genérica;
- G_j = Planta local;
- Sup_j = Supervisor local;
- $RSup_j$ = Supervisor local reduzido.

Observe que a utilização da especificação genérica E_j como supervisor reduzido nem sempre garante que será o menor supervisor, visto que E_j foi modelada pelo projetista. Então, de todo modo foi realizada a redução para verificar o tamanho dos supervisores reduzido gerados pelo TCT.

Para realizar a redução dos supervisores por meio do TCT é necessário utilizar a ferramenta IDES, a qual exporta os autômatos para o TCT e importa-os depois de realizada a redução. O TCT não tem interface gráfica e todas as ações são executadas por comandos. O TCT utiliza para computar a redução dos supervisores, as plantas e os supervisores desenvolvidos pelo IDES. Os seguintes comandos são executados na ferramenta:

- SUPDAT = condat(PLANT, SUPER);
- SIMSUP = supreduce(PLANT, SUPER, SUPDAT).

De todos os supervisores reduzidos gerados pelo TCT, apenas dois tiveram tamanhos diferentes em relação à especificação: $RSup_5 = 2$ estados e $RSup_6 = 2$ estados. A diferença de estados é mínima, então deve ser levado em consideração, que mesmo com mais estados, a utilização da especificação como supervisor reduzido pode ser mais compreensível, a menos que exista erro de modelagem e estados desnecessários foram criados ou que uma solução melhor para a especificação foi encontrada; ou que seja realmente necessário diminuir a quantidade de estados. Para a continuidade da metodologia, optou-se pela escolha dos supervisores reduzidos com três estados.

3.3.3 Emulação

A emulação do funcionamento dos sistemas produtos da planta sob o controle dos supervisores reduzidos obtidos na etapa 3.3.2 foi realizada na ferramenta Supremica, pois a ferramenta IDES não possui essa opção. Foram encontrados diversas vezes problemas de bloqueio e estados proibidos que estavam habilitados, o que necessitou de alteração em alguns autômatos. Os problemas foram sendo corrigidos e a lógica testada novamente. Esta fase foi concluída quando a planta do módulo Distribuição do MPS estava em conformidade com a lógica proposta.

A Figura 23 mostra a emulação da lógica de controle supervisório do módulo Distribuição sendo executada por meio do *software* Supremica.

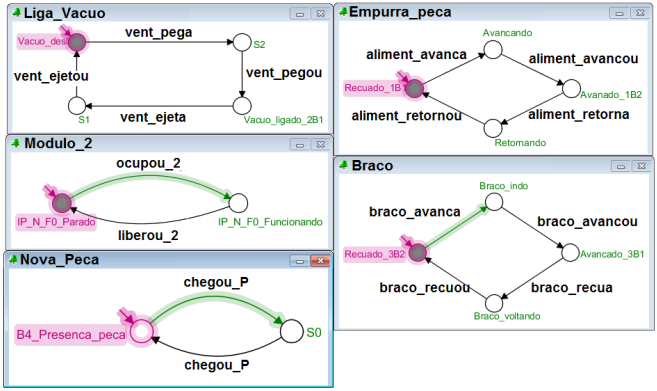

Figura 23 – Tela de Emulação do Controle Supervisório.

3.3.4 Implementação de Controle Supervisório em CLP

Após a síntese de controle supervisório, obtêm-se todos os supervisores modulares locais reduzidos necessários para controlar a planta. Para esse controle ser implementado em CLP é preciso transcrever o código em uma linguagem que seja interpretada pelo CLP. A linguagem utilizada para implementar os supervisores no CLP é a linguagem ST. A ferramenta IDES2ST, desenvolvida por KLINGE (2007) consegue importar os autômatos gerados pelo IDES e gerar automaticamente o código em linguagem ST.

A TCS assume que eventos acontecem espontaneamente e os supervisores desativam eventos controláveis temporariamente até que a planta atinja um estado onde esses eventos podem ocorrer. No entanto, os sinais do sistema físico não são acionados automaticamente pela lógica de controle supervisório, ou seja, os acontecimentos na lógica de controle supervisório não representam diretamente um sinal de entrada ou saída do CLP, eles são apenas uma abstração do comportamento do sistema real. Portanto, sequências operacionais fazem o papel de atualizar os sinais de saída do CLP de acordo com a lógica de controle supervisório e interpretar os sinais de entrada como eventos não controláveis que atuaram na lógica de controle.

A fim de implementar os supervisores gerados no CLP e fazê-los controlar como assumido pela TCS, é implementada uma hierarquia de três níveis, fazendo com que haja uma interface entre os supervisores e o sistema real (QUEIROZ; CURY, 2002b). A implementação segue a

estrutura explicada na seção 2.4.

O código em ST é implementado no software TIA Portal v11 da siemens, o qual possui a ferramenta STEP 7. Essa ferramenta disponibiliza uma linguagem de alto nível SCL (*Structured Control Language*) baseada em PASCAL. Essa linguagem está dentro da norma IEC 61131-3 atendendo os níveis básicos da linguagem ST. Assim, o código do supervisor gerado por meio do IDES2ST é compatível com essa linguagem.

A programação no STEP 7 permite organizar o programa por blocos, os quais são chamados em um programa principal que é executado ciclicamente. Como a estrutura de implementação é dividida em três partes (Supervisores reduzidos, Sistemas Produto e Sequências Operacionais), cada parte é implementada em um bloco diferente. A Fase IV da metodologia é dividida em quatro etapas: (*i*) Implementação dos supervisores modulares e sistema produto; (*ii*) Inicialização e Definição de Modo de Operação; (*iii*) Implementação das sequências operacionais; (*iv*) Teste inicial do funcionamento do sistema. Observa-se que a etapa (*ii*) não se encontra na metodologia apresentada na Figura 6, mas Portilla (2011) implementou essa etapa em seu trabalho, porém, na Fase V da metodologia. No entanto, o autor dessa dissertação achou necessário implementar essa etapa antes das sequências operacionais, devido aos testes realizados durante a implementação do controle supervisório em CLP.

(i) - Implementação dos Supervisores Modulares e Sistema Produto.

O código gerado pelo IDES2ST segue a seguinte estrutura: o código gerado é dividido em dois blocos de função (FB_Supervisores e FB_Sistema_Produto); uma tabela com todas as variáveis globais da lógica é gerada; são criadas variáveis para informar o estado atual de cada supervisor e de cada subsistema; são criadas duas variáveis auxiliares (*ilc_inited* e *evt_blk*).

Ao iniciar a execução da lógica, o programa leva todos os supervisores e subsistemas para seus estados iniciais. Então, a variável *ilc_inited* é setada, o que impede que no próximo ciclo o sistema defina valores iniciais para os supervisores e subsistemas novamente, garantindo a lógica de inicialização apenas no primeiro ciclo.

A variável *evt_blk* é utilizada para indicar que deve haver uma atualização de estado no sistema produto e também nos supervisores. Quando *evt_blk* é ativada, nenhuma ação pode ocorrer até que o sistema seja atualizado e que *evt_blk* seja resetada. O programa gera também

as seguintes variáveis:

- *e_ < evento >*: padrão em que os eventos descritos na Tabela 1 são criados. Exemplo: *e_braco_avanca*;
- *Ae_ < evento >*: essa denotação, representa as variáveis que interpretam os sinais de resposta da planta. Esses eventos ativam os eventos não controláveis da lógica. Exemplo: *Ae_braco_avancou* ativa *e_braco_avancou*;
- *De_ < evento >*: indica que o evento controlável está desabilitado. Exemplo: *De_braco_avanca*;
- *p_ < evento > _ St*: variável que contém o valor do estado atual do sistema produto correspondente. Exemplo: Figura 25;
- *s_ < numero do supervisor > _ St*: variável que contém o valor do estado atual do supervisor correspondente. Exemplo: *s_Sup_Reduzido_6_St*;

A parte do código relacionado aos supervisores é implementada no *FB_Supervisores* e é dividida em duas partes. A primeira implementa os supervisores, já a segunda trata das desabilitações de eventos.

```
(*s_SUP_REDUZIDO_10:*)
"evt_blk" := 0;
IF ((NOT "evt_blk") AND ("s_SUP_REDUZIDO_10_St"=0) AND "e_ocupou_2") THEN
   "evt_blk" := 1;
   "s_SUP_REDUZIDO_10_St":=1;
END_IF;

IF ((NOT "evt_blk") AND ("s_SUP_REDUZIDO_10_St"=1) AND "e_liberou_2") THEN
   "evt_blk" := 1;
   "s_SUP_REDUZIDO_10_St":=0;
END_IF;
(*e_braco_avanca:*)
"De_braco_avanca" := 0;
IF("s_SUP_REDUZIDO_10_St"=1)
THEN
   "De_braco_avanca" := 1;
END_IF;
```

Figura 24 – Código do supervisor número 10.

Podemos ver na Figura 24, o código do supervisor 10, referente a especificação ilustrada na Figura 22. Nota-se que nesse exemplo, apenas o evento *De_braco_avanca* é desabilitado pelo supervisor. A desabilitação ocorre quando o supervisor se encontra no estado 1.

O sistema produto é programado no FB_Sistema_Produto, o qual implementa primeiro a lógica dos eventos não controláveis, devido à sua imprevisibilidade, e depois a lógica dos eventos controláveis. Desta forma, antes que qualquer evento controlável possa ocorrer, o programa verifica os eventos não controláveis e atualiza o sistema caso ocorra alguma mudança.

```
35 IF ((NOT "evt_blk") AND ("p_Braco_St"=1) AND (NOT "De_braco_avanca")) THEN
36    "e_braco_avanca" := 1;
37    "evt_blk" := 1;
38    "p_Braco_St":=2;
39    #LOGCODE := 2;
40 END_IF;
41
42 IF ((NOT "evt_blk") AND ("p_Braco_St"=2) AND "Ae_braco_avancou") THEN
43    "e_braco_avancou" := 1;
44    "Ae_braco_avancou" := 0;
45    "evt_blk" := 1;
46    "p_Braco_St":=3;
47    #LOGCODE := 10;
48 END_IF;
49
50 IF ((NOT "evt_blk") AND ("p_Braco_St"=3) AND (NOT "De_braco_recua")) THEN
51    "e_braco_recua" := 1;
52    "evt_blk" := 1;
53    "p_Braco_St":=4;
54    #LOGCODE := 6;
55 END_IF;
56
57 IF ((NOT "evt_blk") AND ("p_Braco_St"=4) AND "Ae_braco_recuou") THEN
58    "e_braco_recuou" := 1;
59    "Ae_braco_recuou" := 0;
60    "evt_blk" := 1;
61    "p_Braco_St":=1;
62    #LOGCODE := 11;
63 END_IF;
```

Figura 25 – Sequência de eventos no sistema produto do Braço.

Na Figura 25 é mostrado um exemplo de código do sistema produto implementado, referente aos estados do braço. Cada valor da variável *p_braco_St* corresponde a um estado do sistema produto apresentado na Figura 10 e a variável *De_braco_avanca* corresponde ao evento que bloqueia o avanço do braço, representado por *e_braco_avanca*.

(ii) - Inicialização e Definição de Modo de Operação.

Segundo Vieira (2007),

> A definição e divulgação de métodos formais não

> resultam na alteração da prática industrial com relação à implementação do controle de sistemas a eventos discretos, porém, são condições necessárias para tanto.

O mesmo estabelece um método para implementar a Arquitetura de Controle Supervisório, visando superar algumas limitações, como conduzir o sistema para o estado inicial por meio de uma sequência de tarefas pré-estabelecidas, reação em caso de emergência, interrupção da geração de eventos controláveis ou geração seletiva de eventos controláveis.

O método é aplicado em um bloco OB denominado *Main*, que é o bloco principal e que faz a chamada dos outros blocos. Essa lógica é desenvolvida em linguagem LADDER, devido ao fato de ser uma linguagem de mais baixo nível e mais comum em indústrias, além de que a versão do CLP S7 1200 não suporta a Linguagem SFC (Sequencial Function Chart (INTERNATIONAL..., 1998)), a qual é utilizada por (VIEIRA, 2007). A Figura 26 ilustra a estrutura do programa principal adaptado do método de implementação desenvolvido por Vieira (2007).

O método é composto por seis modos distintos de operação.

- *Software Inicialization* (SI);
- *Physical System Initialization* (PSI);
- *Manual* (Man);
- *Supervised* (Sup);
- *Emergency* (Emg);
- *Idle* (ocioso), que abrange Init e PSIted;

Considera-se que o passo *init* é ativado na primeira execução do programa, o que faz setar a variável *Sinit* automaticamente, resultando na ativação do modo *SI*. Neste modo, é zerado o valor de todas as variáveis internas do CLP.

Para passar para o modo *PSI* é necessário setar duas variáveis. A primeira é a variável interna *PSinit*, que é acionada automaticamente após a finalização do modo *SI*. A segunda variável é uma entrada física do CLP, referente a um botão *Reset* instalado na IHM. Entrando no modo *PSI*, o sistema realiza uma sequência de eventos que o levam para o estado inicial da máquina. Concluída a inicialização do sistema, a variável interna *PSready* é ativada.

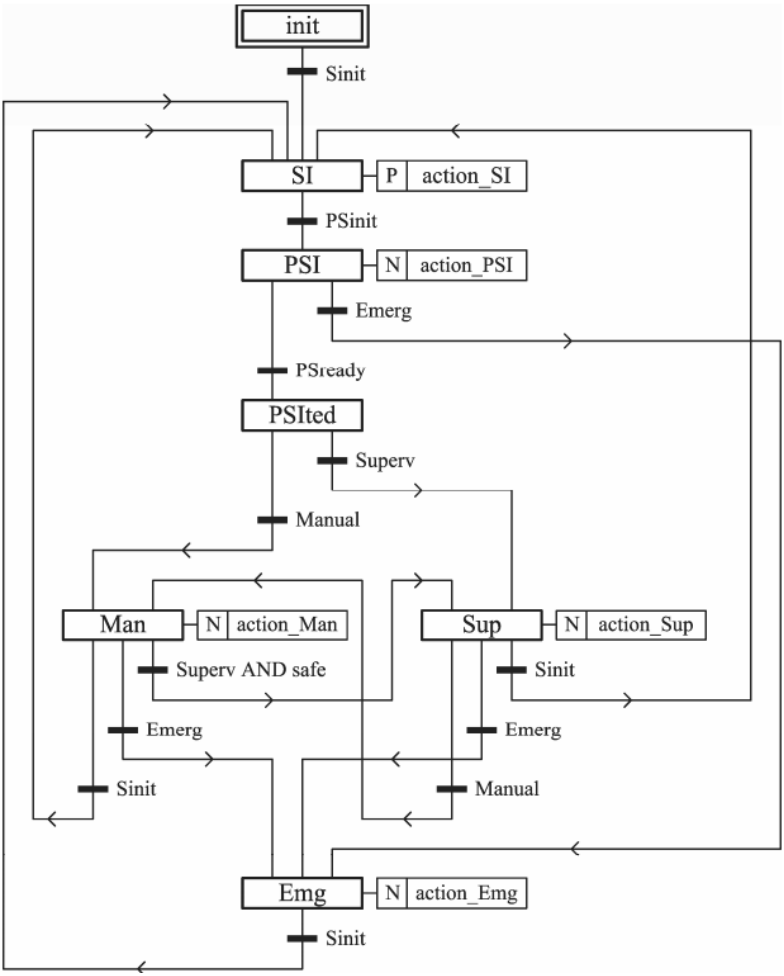

Figura 26 – SFC Main(VIEIRA, 2007)

Para entrar no modo *PSIted* é necessário que além da variável *PSready*, a entrada física do CLP referente ao botão *Start* da máquina também seja ativa. Nesse modo o sistema fica *idle* até que seja definido o modo de operação (Manual ou Supervisionado).

No caso do módulo Distribuição, existe apenas um botão com duas posições físicas para a escolha do modo, localizado na IHM do módulo. Em uma posição do botão o CLP recebe 1 (um) e em outra

0 (zero). Em cima disso, é definido que quando o CLP recebe 0, o sistema entra no modo *Sup*, onde a coordenação das ações a serem realizadas é definida conforme o estado ativo dos supervisores. Se o CLP recebe valor 1, o sistema entra no modo Man, no qual o operador é responsável por conduzir o acionamento dos eventos. Neste modo de operação, todos os sinais de desabilitação são ativados, o que inibe a geração de qualquer evento controlável.

É implementada no modo *Man* a possibilidade de realizar as transições de forma passo-a-passo, onde a cada clique no botão *Start* ocorre o evento que está habilitado pelo controle supervisório, fazendo com que o sistema evolua um estado a cada clique. Este modo de evolução ajuda na verificação da modelagem.

É possível mudar do modo *Man* para o modo *Sup*, desde que a lógica do CLP esteja sincronizada com a lógica de controle supervisório. A operação inversa é realizada sem restrições.

O último modo é o *Emg*, que é ativado pressionando o botão *Stop* da máquina. Neste modo todas as atividades são suspensas imediatamente. Após esse modo ser finalizado, a variável *Sinit* é ativada automaticamente, o que faz retornar para o modo *SI*, onde deve ser realizado novamente todo o processo de inicialização do sistema.

Não é implementada a transição direta dos modos *Man* e *Sup* para o modo *SI* por meio da variável *Sinit*. Isso se deve ao fato de não ter um botão apropriado no painel do módulo Distribuição e também porque sendo acionado o botão *Stop* o sistema retorna para o modo *SI* da mesma forma.

(iii) - Implementação de Sequências Operacionais.

Após a implementação dos sistemas produtos e dos supervisores no CLP, utilizando a linguagem ST, são definidas as sequências operacionais em linguagem LADDER, as quais traduzem os comandos (eventos controláveis) do sistema produto para gerar os sinais de saída do CLP e interpretam os sinais que chegam nas entradas do CLP como respostas da planta (eventos não controláveis).

Ao ser ativada a variável $e\_ < evento >$ pelo sistema produto, as sequências operacionais devem implementar sub-rotinas para ativar a saída do CLP correspondente ao evento ativado. Após concluir a ação, a planta envia um sinal para entrada do CLP, que deve executar uma outra sub-rotina para setar a variável $Ae\_ < evento >$ correspondente. Essa variável ativa o evento não controlável $e\_ < evento >$ correspondente.

A Figura 27 mostra como exemplo o acionamento do avanço do

braço. O sistema produto atualiza as informações sobre os estados do subsistema do braço e o supervisor desabilita os eventos que não podem ocorrer. Caso o evento *De_braco_avanca* não esteja habilitado e o braço se encontre na posição recuada (valor igual a 1, conforme representado na Figura 10), o evento *e_braco_avanca* é executado. Então, as sequências operacionais acionam a saída física do CLP $\%Q0.4$, fazendo com que o braço avance. O braço começa a rotacionar até chegar na posição P2, onde está localizado o sensor fim-de-curso, o qual deve enviar o sinal na entrada $\%I0.5$ indicando a chegada do braço. O CLP recebe esse sinal, executa as sequências operacionais e aciona a memória interna $\%M37.0$ (memória relacionada ao evento *Ae_braco_avancou*). Esse sinal é interpretado pelo sistema produto como evento não controlável.

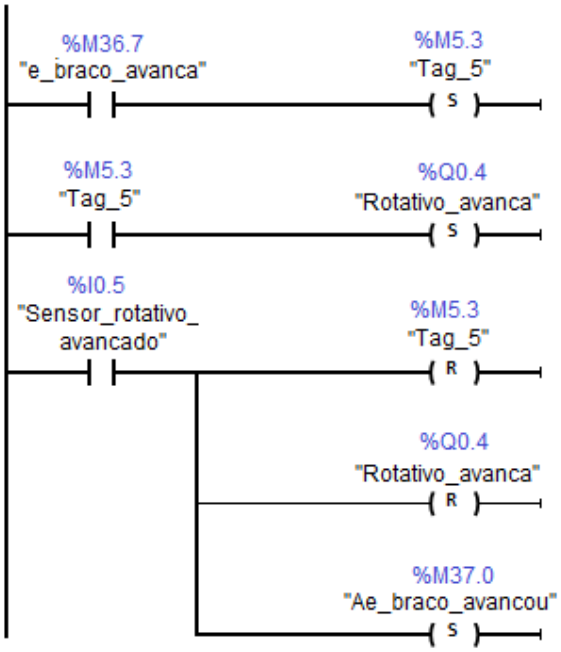

Figura 27 – Exemplo Sequência Operacional - Sistema Produto "Braco".

(iv) - Teste Inicial do Funcionamento do Sistema.

Nessa etapa é realizada uma verificação para encontrar possíveis

erros na lógica. Esta verificação pode não ser realizada de forma exaustiva.

O teste inicial identificou logo no início o primeiro problema, apenas o braço se movia de um lado para outro. O programa sempre escolhia o evento de avançar o braço, fazendo que o evento "avançar alimentador" nunca fosse executado, apesar de também estar habilitado pelo supervisores. A preferência pela escolha dos eventos relacionados ao braço se deve pela posição em que se encontram na lógica de programação do CLP, ou seja, um evento programado na linha de código 25 do CLP executa antes de um evento programado na linha 32. Esse problema só ocorre quando dois ou mais eventos podem ocorrer em um mesmo ciclo de varredura da lógica do CLP.

Para solucionar o problema, optou-se por retornar para a fase 2.2 e acrescentar uma nova especificação, gerando um novo supervisor modular local reduzido. A especificação adicionada foi a de número 8 (E_8_$RecuoBraco$), a qual evita que o braço avance ou recue sem necessidade.

Outro problema encontrado foi em relação ao avanço do braço e a ativação da ventosa. Caso a ventosa iniciasse a ação de pegar a peça e, antes que ela concluísse a ação, o braço acionasse para avançar, a peça poderia não estar completamente presa e acabava se soltando. Para resolver esse problema é definida uma nova especificação, a qual determina que após ocorrer o evento *vent_pega*, o braço não avança enquanto não ocorrer o evento *vent_pegou*, indicando que a peça está completamente presa. A especificação adicionada para resolver esse problema é a de número 6 (E_6_$Braco_OU_Ventosa$).

Nesse fase, verificou-se a eficiência da síntese de supervisores usando a abordagem modular local. Para adicionar uma nova especificação não foi necessária uma remodelagem, apenas foi acrescentada uma nova especificação e gerado um novo supervisor local. Nenhum evento novo foi adicionado.

3.3.5 Implementação de Funcionalidades Básicas do Sistema SCADA

Nessa seção, são apresentadas as implementações da Fase V da metodologia para o desenvolvimento de um sistema SCADA, subdivididas em duas etapas: (*i*) Instalação e Configuração do *Software* SCADA; (*ii*) Implementação de Funcionalidades Básicas. O objetivo dessa fase é desenvolver um sinótico que represente toda a planta e, que a mesma

possa ser operada e visualizada pelo operador através desse sinótico.

(i) - Instalação e Configuração do Software SCADA.

O *software* TIA Portal da Siemens integra a ferramenta WinCC, a qual realiza o desenvolvimento do sistema SCADA, mas essa ferramenta é adquirida e instalada separadamente. No TIA Portal é necessário adicionar um dispositivo para ser o servidor SCADA. O modelo de dispositivo utilizado para essa aplicação é o *SIMATIC PC-Station* versão 1.0. Então, no PC-Station é adicionado o WinCC, modelo WinCC RT Professional, para desenvolvimento da parte gráfica. Em seguida é estabelecida a conexão entre o CLP e o *PC-Station* utilizando o protocolo PROFINET para comunicação entre os dispositivos. Por fim, é verificado o envio de dados entre as duas ferramentas como teste de comunicação. Essas configurações são realizadas apenas uma vez, visto que é utilizado sempre o mesmo computador como servidor.

(ii) - Implementação de Funcionalidades Básicas.

A implementação é iniciada com o desenvolvimento das representações gráficas da planta. Todas as representações estão relacionadas com as variáveis criadas pelo controle supervisório. Essas variáveis são mapeadas para o WinCC, onde são conectadas às ilustrações da planta. A tela que contém essas informações é chamada de sinótico, que é uma representação gráfica e textual de todas as informações sobre a planta necessárias para o operador.

O sinótico que representa o funcionamento da planta utiliza os valores dos estados dos autômatos do sistema produto, representados pela variável $p\_ < evento > \_St$. Cada sistema produto é representado conforme modelado no Supremica e IDES, ou seja, se a planta tem quatro estados, a tela do SCADA representará os quatro estados. Como exemplo, a Figura 28 mostra o sistema produto referente ao vácuo, o qual possui quatro estados, com as respectivas imagens:

As imagens apareceram no SCADA conforme o valor da variável $p\_Liga\_vacuo\_St$, sendo:

Estado 1 : sem peça;

Estado 2 : pegando uma peça;

Estado 3 : com a peça pega;

Estado 4 : soltando a peça.

As imagens foram implementadas seguindo a estrutura que leva em conta o valor da variável $p\_ < evento > \_St$. Porém, houve dois

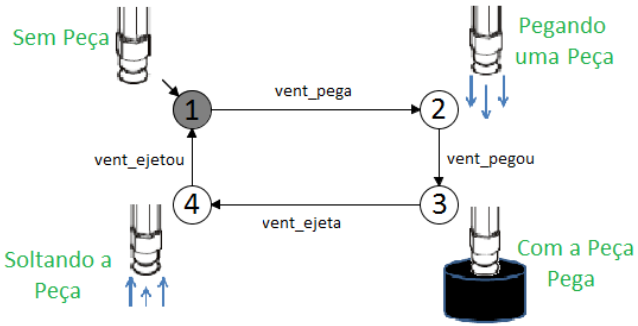

Figura 28 – Exemplo dos estados da ventosa

casos em que se fez necessário realizar adaptações: sistema produto referente a chegada de peças e sistema produto referente ao acionamento da ventosa.

No caso do sistema produto $SP_4\_Ventosa$, ilustrado na Figura 11, a ventosa se move junto com o braço, o que faz com que ela possa estar tanto na posição P1 quanto na P2, ou ainda em uma posição intermediária entre P1 e P2, ou seja, somente a informação do estado do seu sistema produto não serve para representar a posição da ventosa. Devido a isso, utiliza-se além do valor do estado do sistema produto $SP_4\_Ventosa$, o valor do estado do sistema produto SP_3_Braco, indicando desta forma, a posição e o estado da ventosa .

É realizada a equação 3.1 para calcular a posição e o estado da ventosa. O resultado é atribuído a uma variável chamada *vacuoEbraco*. O resultado do cálculo tem a dezena indicando a posição do braço e o valor da unidade, o estado em que a ventosa se encontra.

$$vacuoEbraco := p\_Liga\_vacuo\_St + 10 \times p\_Braco\_St \tag{3.1}$$

No caso do sistema produto $SP_1\_NovaPeca$, representado na Figura 8, as informações do estado não são suficientes para representar a posição das peças e o percurso delas, devido a isso, é necessário utilizar as informações dos supervisores. A peça pode estar em uma de duas posições (posição de chegada (Pc) ou P1), quando a peça está na posição P1 pode chegar outra peça na posição Pc. Para representar que há peça na posição pc, utiliza-se o valor do estado do supervisor $s\_Sup\_Reduzido\_6\_St$ (valor deve ser igual a 1) e do estado do sistema produto $p\_Empurra\_Peca\_St$ (valor deve ser igual a 1). Para indicar

que há uma peça na posição P1, utiliza-se apenas o valor do estado do supervisor *s_Sup_Reduzido_1_St*, que nesse caso deve ser igual a 1.

Comandos remotos também são disponibilizados no sinótico. Todos os botões da IHM do módulo Distribuição são implementados no SCADA para acionamento remoto, exceto o botão físico que define o modo de operação entre manual e automático. Isso se deve ao fato de ser um botão com chave e posição fixa (os demais são todos botões de pulso). Esse botão com chave é ilustrado como imagem para indicar a posição atual da chave. Os botões *Start*, *Reset* e *Stop* estão ligados à variáveis internas do CLP, criadas para que quando ativadas simulem a ação de pressionar os botões da IHM, realizando assim um acionamento remoto. Também são disponibilizados botões para forçar a ocorrência de eventos, para cada evento controlável existe um botão "*Force*".

No modo *Man* os eventos ocorrem de duas maneiras: conforme é pressionado o botão *Start*, evoluindo de forma passo-a-passo; ou forçando os eventos que estão elegíveis (eventos que estão habilitados pelo supervisor). Para forçar o ocorrência de um evento é necessário clicar sobre o botão referente à ação desejada. Os botões só poderão ser acionados quando em modo *Man* e estando elegível.

As três situações possíveis em que os eventos podem se encontrar são diferenciadas na tela do SCADA. Os eventos que são desabilitados e ficam invisíveis, os eventos elegíveis são indicados com uma cor verde, e os eventos que são fisicamente possíveis de ocorrer, porém estão desabilitados, são indicados por uma cor vermelha. Para implementar a possibilidade de forçar os eventos, é alterado o código do sistema produto relacionado aos eventos controláveis, já que a modelagem por controle supervisório não prevê essa situação.

Como exemplo de alteração de código para implementar a possibilidade de forçar eventos, pegamos o código referente à ação de avançar o braço. Originalmente, o código depende de três variáveis para executar *e_braco_avanca*: a que indica o valor do estado (valor deve ser igual a 1), a que indica se o evento está desabilitado (valor deve ser igual a zero) e a que indica atualização de estado no sistema produto (valor deve ser igual a zero).

Com a alteração, o valor do estado e a confirmação de que não há atualização continuam tendo que ser satisfeitas. Porém, mais duas condições foram adicionadas, e pelo menos uma tem que ser satisfeita. As duas condições são:

Condição 1 : o evento não pode estar desabilitado e a variável *enable_controllable_events* tem que ser verdadeira; ou

Condição 2 : modo manual selecionado e variável *force_braco_avanca* habilitada.

Alguns eventos podem ocorrer devido ao estado atual do sistema produto, porém podem estar desabilitados pelos supervisores. Em caso do operador forçar a ocorrência de algum desses eventos, a lógica vai para uma situação de bloqueio e o sistema não estará mais executando em segurança. Nesse caso, fica impossibilitada a ação de mudar do modo *Man* para o modo *Sup* e a variável *unsafe* é habilitada fazendo com que apareça na tela do sinótico um aviso de alerta. Para corrigir a situação de insegurança, é necessário reiniciar.

Relatório de eventos:

Em algumas situações é necessário saber se um evento ocorreu ou não, ou ainda saber quais os últimos eventos que ocorreram, principalmente para análise de problemas. O relatório implementado para o módulo Distribuição registra todos os eventos controláveis e não controláveis que acontecem, de forma ordenada, para que seja possível, por exemplo, conhecer a sequência de eventos ocorridos até o momento de um erro na lógica.

A implementação é realizada em três partes: (*a*) Incremento dos eventos na fila; (*b*) Retirada de eventos da fila; (*c*) Implementação no SCADA.

O relatório é criado em forma de fila, onde a cada ocorrência de um evento é armazenada uma variável, referente a esse evento, em uma posição na fila. Conforme é registrado um evento da fila no relatório de eventos, o ponteiro muda a posição da fila para a próxima posição a ser registrada.

A Tabela 5 descreve as variáveis utilizadas para implementar o relatório de eventos.

Tabela 5 – Descrição dos Eventos utilizados

Variável	**Descrição**
eventos	armazena o nome do evento que ocorreu
num_eventos	armazena o número de eventos que ocorreram
num_eventos_lido	armazena o número de eventos já registrados
índice	número do índice do *array*
evento_lido	próximo evento a ser registrado

(*a*): Um *array* de 100 posições (0 a 99) é criado para armazenar os eventos que vão ocorrendo. Para todos os eventos controláveis e

não controláveis são criadas variáveis constantes, com valores fixos e que representam aquele evento. No sistema produto quando ocorre um evento, uma variável recebe a constante relacionada ao evento que ocorreu, e além disso, a variável *evt_blk* recebe 1 (um), indicando uma atualização no sistema.

No bloco de lógica referente aos supervisores é implementada a fila de relatórios. Quando tem uma atualização no sistema, indicada pela variável *evt_blk*, a lógica incrementa mais 1 ao valor da variável *num_eventos*. Essa variável indica a posição da fila em que a constante referente ao evento deve ser armazenada. Quando chegar a posição 99 do *array*, a contagem é reiniciada. Então, a variável *evt_blk* é zerada para que um novo evento possa ocorrer. Na Figura 29 é apresentado o código referente ao armazenamento dos eventos na fila. A fila inicia da posição 1, mas após a posição 99, ela reinicia da posição 0.

Código	Comentário
`IF (NOT "evt_blk") AND ("p_Empurra_Peca_St"=1)`	
`  AND (NOT "De_aliment_avanca") Then`	`lógica verdadeira`
`  "e_aliment_avanca" := 1;`	`variável recebe 1`
`  "evt_blk" := 1;`	`variável recebe 1`
`  "eventos":= "c_aliment_avanca";`	`"eventos" recebe a constante referente a esse evento`
`END_IF;`	
`IF ("evt_blk") THEN`	`lógica verdadeira`
`  "evt_blk":= 0;`	`variável recebe 0`
`  "num_eventos" := "num_eventos" + 1;`	`0 + 1 = 1 - valor da variável`
`  IF "num_eventos" < 100 THEN`	`lógica verdadeira`
`    "i" := "num_eventos";`	`"i" recebe 1 - posição atual em que deve ser armazenada a variável`
`  ELSE`	`lógica falsa`
`    "i" := "num_eventos" MOD 100;`	
`  END_IF;`	
`  Array["i"] := "eventos";`	`Array[1] - posição 1 do array`
`END_IF;`	

Figura 29 – Código de incremento de eventos na fila.

(*b*): Todos os eventos que estão na fila devem ser registrados no relatório de eventos. A variável *num_eventos_lido* armazena a última posição da fila que foi registrada no relatório. Então, a cada ciclo é verificado se *num_eventos* é maior ou igual a *num_evetos_lidos*. Caso o valor de *num_eventos* seja maior, a variável *índice* recebe o valor do número de eventos que já foram lidos. Por fim, a variável *evento_lido* recebe a variável armazenada na fila, indicada pela variável *índice*.

Caso o valor de *num_evetos_lidos* seja maior ou igual a 100, é realizada uma divisão por 100 e o índice recebe o resto da divisão (função MOD 100). A Figura 30 mostra o código implementado para retirada de eventos da fila.

(*c*): No SCADA é apresentado o relatório de eventos para o

Código	Comentário
IF ("num_eventos">="num_eventos_lido") THEN	lógica verdadeira
IF "num_eventos_lido" < 100 THEN	1 < 100
"indice" := "num_eventos_lido";	indice := 1
ELSE	
"indice" := "num_eventos_lido" MOD 100;	----
END_IF;	
"evento_lido" := Array["indice"];	envento_lido := Array[1]
END_IF;	

Figura 30 – Exemplo do código de incremento de eventos na fila.

operador. Cada vez que o SCADA lê a variável *evento_lido*, a qual armazena o nome da variável que ocorreu, ele escreve o valor dessa variável (o valor é uma constante com o nome do evento que ocorreu) no relatório de eventos e incrementa mais 1 ao valor da variável *num_eventos_lido*, para que no próximo ciclo seja lido o índice seguinte do *array*. Por fim, a variável *evento_lido* recebe valor 0 (zero).

3.3.6 Avaliação de Funcionamento do Sistema Real

Após a implementação do sistema SCADA, é realizado o teste de avaliação da planta. Os testes incluem a visualização da planta em tempo real, o envio de comandos remotos e o desempenho completo da lógica de controle supervisório. Nesta fase todos os dispositivos de campo já estão conectados ao CLP.

Durante a avaliação, uma situação que causa problema, mas que não havia sido especificada e que nos testes anteriores não ocorria, foi encontrada. A Figura 31 mostra a sequência que levava ao problema.

Após o alimentador avançar para levar uma peça à posição P1, dois eventos eram habilitados: alimentador recuar ou braço recuar (estado $S1$ da Figura 31). Caso a escolha fosse de recuar o braço, a lógica mudava para um estado onde uma de duas possibilidades de eventos, poderia ocorrer (estado $S2$ da Figura 31): alimentador recuar ou ventosa acionar. Caso a ventosa fosse acionada, a lógica evoluía de estado e dois eventos eram habilitados (estado $S4$ da Figura 31): recuar o alimentador ou avançar o braço. Se nesse momento o braço avançasse, evoluindo para o estado $S6$ da Figura 31, um problema ocorria. Como o alimentador estava avançado, ele mantinha a peça presa, o que acabava fazendo com que a peça escapasse da ventosa, em um estado não permitido.

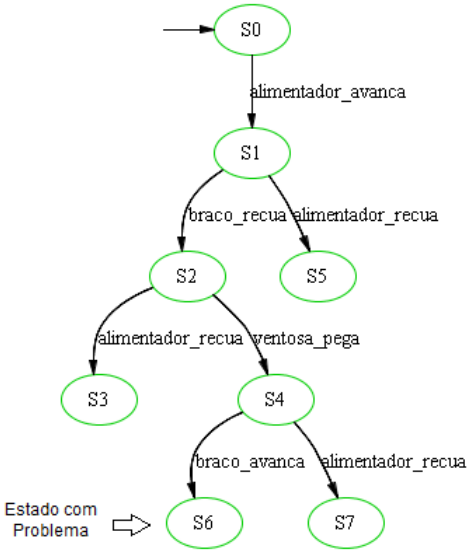

Figura 31 – Exemplo da Sequência que leva para o problema.

Para corrigir o problema, foi retornado para a fase 3.3.2 e modelada a especificação E_7_$EvitarPecaPresa$, que determina que a ventosa não ativa se o alimentador não estiver recuado. Com essa especificação, foi gerado um novo supervisor local, o qual foi implementado junto ao código já no CLP. Esse supervisor, diferente dos anteriores, foi implementado manualmente no código, inserindo o supervisor e as desabilitações necessárias. Dessa forma, não foi necessário gerar um novo código com a ferramenta IDES2ST. Isso só foi possível graças a utilização de supervisores modulares locais.

Para implementar manualmente esse novo supervisor, foi realizada a modelagem da nova especificação no IDES, gerado um novo supervisor e encontrado o supervisor reduzido. Esse supervisor contém dois estados e foi implementado manualmente no *FB_Supervisores*, como pode ser visto na Figura 32. Para saber quais eventos são bloqueados por esse supervisor, é utilizada a ferramenta TCT, a qual gera um arquivo *Condat* que identifica os eventos habilitados em cada estado do supervisor. Foi identificado apenas um estado com evento bloqueado: o evento *vent_pega* desabilitado no estado 1.

O problema não havia sido descoberto nos testes anteriores, de-

```
(*s_SUP_REDUZIDO_7 :*)
"evt_blk" := 0;
IF ((NOT "evt_blk") AND ("s_SUP_REDUZIDO_7_St"=0) AND "e_aliment_avancou") THEN
  "evt_blk" := 1;
  "s_SUP_REDUZIDO_7_St":=1;
END_IF;

IF ((NOT "evt_blk") AND ("s_SUP_REDUZIDO_7_St"=1) AND "e_aliment_retornou") THEN
  "evt_blk" := 1;
  "s_SUP_REDUZIDO_7_St":=0;
END_IF;

(*e_vent_pega:*)
"De_vent_pega" := 0;
IF(("s_SUP_REDUZIDO_1_St"=0)OR("s_SUP_REDUZIDO_2_St"=1)OR("s_SUP_REDUZIDO_4_St"=1)
 OR("s_SUP_REDUZIDO_7_St"=1)  THEN
  "De_vent_pega" := 1;
END_IF;
```

Figura 32 – Código do Supervisor Reduzido 7 implementado manualmente.

vido ao fato de não ter os elementos físicos conectados ao CLP e os modos de operação foram implementados depois dos testes do Supremica. Esse problema foi descoberto graças ao modo passo-a-passo, implementado no modo manual, onde a cada situação era possível escolher o evento a ser executado. Outro ponto que ajudou a identificar e resolver o problema foi o relatório de eventos. Após os ajustes, o sistema encontrou-se em conformidade com o inicialmente planejado.

3.3.7 Implementação de Funcionalidades Gerais do Sistema SCADA

Nesta fase são implementadas funcionalidades gerais para o sistema SCADA, as quais devem complementar o SCADA para facilitar o trabalho do operador. Essas funcionalidades são implementadas conforme as necessidades da empresa. Para esse trabalho foi implementada apenas a geração de alarmes. Outras funcionalidades que podem ser implementadas são: receitas, gráficos de tendências, relatórios gerais (históricos e alarmes), criação de banco de dados com informações sobre a produção, etc. Observa-se que um relatório de eventos foi implementado na Seção 3.3.5, devido ao fato de ajudar nos testes e identificação de problemas.

Os alarmes são mostrados na tela do sinótico para visualização do operador. Abaixo segue a lista de alarmes implementados.

- Unsafe: Indica que o sistema não está mais trabalhando dentro da lógica de controle supervisório;
- Emergência pressionada: Apesar de não ter um botão de emergência instalado, foi definido que quando pressionado o botão *Stop*, um alarme indicando emergência pressionada é mostrado no sinótico;
- Tempo esgotado de trabalho da ventosa: caso a ventosa acione e, por mais de 10 segundos não receba o sinal de peça presa, o alarme ocorre;
- Sem peça: após 5 segundos sem sinal no sensor que indica chegada de nova peça, o alarme é acionado (esse alarme foi definido como classe de "atenção"devido ao fato de não causar problema ao sistema).

3.3.8 Validação

A última etapa do desenvolvimento integrado de sistemas SCADA com controle supervisório é a realização dos testes finais, os quais validam o funcionamento correto do sistema. O módulo Distribuição foi submetido a diversos testes e ciclos de produção. Todos os testes obtiveram resultados satisfatórios, determinando assim, que a coordenação entre os subsistemas está de acordo com as leis de controle definidas e que o sistema SCADA executou com sucesso todas as funcionalidades implementadas.

4 PROPOSTA DE ARQUITETURA DE DESENVOLVIMENTO DE SISTEMA DE CONTROLE SUPERVISÓRIO INTEGRADO AO ROTEAMENTO DE TAREFA

Em alguns casos, a lógica de controle desenvolvida utilizando a TCS, permite que existam situações em que mais de um evento possa ocorrer. Exemplos dessa situação podem ser vistos tanto na Seção 3.3.4 quanto na Seção 3.3.6. A escolha de qual evento deve ocorrer, normalmente se dá por prioridade de implementação no CLP, ou seja, aquele que aparece primeiro no código do CLP executa antes. Este trabalho tem como proposta, incorporar à arquitetura de controle supervisório modular um sistema que faça a escolha desse evento com base em algum requisito ou alguma prioridade.

4.1 O ROTEAMENTO DE TAREFAS

Nesta seção é apresentado o formalismo do roteamento de tarefas, a definição de quem deve fazer esse roteamento, quem deve definir os critérios utilizados para realizar esse roteamento, e como esse roteamento pode interagir com a lógica utilizando a TCS. Todos esses pontos são importantes para o desenvolvimento da metodologia, que é o objetivo desse trabalho.

4.1.1 Sistema de Execução da Manufatura

Para implementar o roteamento das tarefas, é dado destaque ao nível 4 da pirâmide da automação (ver Figura 1), o qual é responsável por integrar as informações que ocorrem no processo com os sistemas que tomam as decisões sobre a produção.

Nesse cenário está o MES, que fornece acesso a todas as informações pertinentes sobre a produção de uma empresa. Ele supervisiona os sistemas de controle de processo, decide sobre as rotas que os produtos seguem através da planta fabril, e também decide quando e onde as produções devem iniciar (VALCKENAERS; BRUSSEL, 2005).

Segundo Bo, Zhenghang e Ying (2004), o MES transfere os dados de produção dos níveis inferiores (niveis 1,2 e 3) para o nível superior (nível 5), com o intuito de otimizar os processos de produção de toda a empresa através da integração das informações. Bo, Zhenghang e Ying

(2004) citam ainda que uma implementação bem sucedida do MES na produção, traz grandes benefícios para as empresas, e que países como os Estados Unidos e Japão, já reconhecem a importância da pesquisa e desenvolvimento de sistemas de MES.

Em 1997 a *Manufacturing Enterprise Solutions Association* (MESA) apresentou a seguinte definição para MES:

> Sistemas de Execução de Manufatura gereciam informações que envolvem a otimização de atividades de produção desde o início de uma ordem de produção até os produtos acabados (ASSOCIATION, 1997).

De uma forma geral, o MES permite ter uma maior flexibilidade em sua fabricação, fornecendo um acompanhamento do planejamento de produção e do estado atual da planta fabril.

4.1.2 Planejamento e Controle da Produção

No último nível da pirâmide da automação está o setor responsável por todos os recursos da empresa. Esse nível é responsável por toda a cadeia de produção e é comumente chamado de nível Enterprise Resource Planning (ERP). O nível ERP pode ser definido como a área de gerenciamento de negócios, podendo ser utilizado para executar, gerenciar e integrar todas as funções dentro de uma organização (MOLINA, 2007).

Dentro desse nível está localizado o setor da empresa responsável pelo PCP ou também referenciado como Planejamento, Programação e Controle da Produção (PPCP). O PCP é responsável pela coordenação e aplicação dos recursos produtivos, de modo a atender da melhor forma possível os planos estabelecidos nos setores estratégicos, táticos e operacionais da empresa (LUSTOSA; MESQUITA; OLIVEIRA, 2008). O PCP vai determinar: o que vai ser produzido; quanto vai ser produzido; onde vai ser produzido; como vai ser produzido; e quando vai ser produzido.

Segundo Gangneux (1989), os sistemas PCPs são necessários porque a capacidade de produção deve ser planejada para atender à demanda especificada pelo *marketing*. A eficácia do PCP pode ser avaliada pelo alcance dos objetivos de redução dos *lead times* (ou ciclo) de produção, dos custos de estoques (matéria-prima, materiais em processo e produtos acabados), de produção (ociosidade, horas extras, etc) e cumprimentos de prazos e agilidade de resposta diante de alteração de demanda (MESQUITA; CASTRO, 2008).

4.1.3 Sistema de Roteamento de Tarefas

O roteamento de tarefas é uma funcionalidade do MES responsável por decidir sobre qual comando deve ser executado pelo CLP em função das estratégias de produção definidas pelo PCP. Desta forma, a escolha dos caminhos de produção pode mudar conforme as necessidades do PCP.

Para esse trabalho, divide-se a lógica de controle entre os níveis hierárquicos CLP e o SRT. O CLP coordena os sensores e atuadores da planta de acordo com as especificações locais de segurança em uma lógica de controle minimamente restritiva e não-bloqueante. Assim, o nível do CLP assegura que os eventos que acontecem na planta são sempre não-bloqueantes e seguros. O nível SRT é responsável pela escolha do próximo evento a ser executado na planta, a qual é realizada de acordo com o layout da planta, do conjunto de eventos elegíveis e de informações oriundas do PCP.

O funcionamento do SRT resume-se em ler informações sobre os eventos e decidir qual comando deve ser executado dentre aqueles que estão habilitados pelo controle supervisório, de modo que a planta execute com eficiência as tarefas definidas pelo PCP. O SRT pode também realizar um controle de demanda de produção de peças, produzindo apenas a quantidade definida pelo PCP em cada pedido. A cada troca de pedido, uma nova demanda de produção é definida e as definições sobre as escolhas dos comandos são atualizadas. É importante que o SRT interaja constantemente com o controle supervisório implementado no CLP, a fim de saber quais eventos estão elegíveis.

Na literatura é encontrado um trabalho semelhante, onde Molina (2007) desenvolve um Coordenador de Roteamento com base na TCS e nas Redes de Petri Coloridas (RPC) para sistemas de manufatura. A TCS é utilizada para tratar requisitos de controle e as RPC são utilizadas para tratar o roteamento dos produtos no sistema de manufatura. A ideia é que o controle supervisório receba ordens do coordenador de roteamento contendo a sequência de fabricação, as quais são utilizadas pelo sistema produto para realizar desabilitações adicionais de eventos controláveis. O sistema ERP estabelece diferentes roteiros de produção possíveis para diferentes tipos de produtos. O coordenador de roteamento recebe essas informações e então converte-as em fichas que trafegam pela rede. Cada ficha na rede identifica um único produto, sendo que em cada ficha é definida a sequência de produção. Para saber mais sobre Redes de Petri ver Cardoso e Valette (1997), Maciel, Lins e Cunha (1996).

Silva et al. (2011) implementam uma forma para lidar com o roteamento em uma célula de manufatura automatizada real. Os autores decompõem o problema de controle em um conjunto de especificações. Quando ocorrem mudanças no *layout* do sistema, ou nas operações atribuídas às estações de trabalho ou mesmo nas regras de produção, é necessário modificar os autômatos que representam as especificações associadas aos subsistemas afetados. Essas mudanças afetam a síntese de controle, o que implica em refazer a síntese dos supervisores locais, cujas especificações foram modificadas. Os demais supervisores não são afetados.

Existem outros métodos estudados para uma solução mais simples, onde a escolha é feita de forma aleatória e sem uma interação com o nível gerencial da empresa. A seguir, são apresentados alguns métodos.

Pinotti, Leal e Oliveira (2010) propõem uma solução para a escolha de eventos em situações de múltipla escolha. O método utiliza uma abordagem em que a escolha é realizada de forma aleatória, o que possibilita a seleção de qualquer um dos eventos que estão habilitados. A solução consiste em um contador que é incrementado a cada ciclo de varredura e o limite da contagem é o número de eventos envolvidos em cada estado. A escolha do evento é realizada conforme o valor do contador. É necessário um contador para cada quantidade de eventos envolvidos. Esta solução exige que o programador conheça todos os estados em que haverá problema de múltiplos eventos habilitados.

Outro método de escolha é visto em CRUZ et al. (2009), onde é definida uma variável de memória que a cada ciclo de varredura assume um valor diferente (0 ou 1), ou seja, se o valor da memória for 0, "A"executa, quando o valor for 1,"B"executa, o que garante uma escolha intercalada dos eventos. Para casos de mais de dois eventos habilitados em um mesmo estado o autor sugere a utilização de uma memória numérica. Esta solução também exige o conhecimento de todos os casos de dois ou mais eventos habilitados no mesmo estado.

Sebem e Leal (2013)apresentam uma proposta para solução do problema, utilizando uma variável *Random*, do tipo inteira, que é incrementada a cada ciclo de varredura do CLP (de 0 a um valor máximo). Quando ocorrer o problema da escolha, o controlador deve escolher 1 entre n (o número de eventos habilitados). O resto da divisão inteira de *Random* por n, retornará um valor entre 1 e n, que será o resultado da escolha. Este método permite que todos os eventos possam ser escolhidos em algum momento, mas não garante que todos serão.

4.2 ARQUITETURA PARA INTEGRAÇÃO DO ROTEAMENTO AO SISTEMA DE CONTROLE SUPERVISÓRIO;

No Capítulo 3 é apresentada a metodologia de Portilla (2011) para desenvolvimento integrado de sistemas SCADA com a programação de controle supervisório em CLP utilizando a TCS. Mas essa metodologia não explora o grau de liberdade de um sistema de controle supervisório e a escolha acaba sendo feita pelo CLP, onde a ordem de implementação das linhas de código é quem determina a escolha do evento. Além disso, a metodologia apresentada trabalha apenas nos três primeiros níveis da pirâmide da automação (ver Figura 1), não interagindo com os níveis gerenciais da empresa.

Um dos principais objetivos desse trabalho é explorar esse grau de liberdade e escolher o caminho com base em informações fornecidas. Com base nisso, é estudado a integração da metodologia desenvolvida por Portilla (2011) com os níveis superiores da pirâmide da automação (4 e 5), a fim de que o PCP possa definir os comandos a serem executados pelo CLP. Nesta seção propõe-se definir o sistema de controle supervisório dentro dos níveis da pirâmide da automação, definir uma arquitetura de sistema de controle supervisório que adapta um SRT, além de um sistema que gerencie os modos de operação.

4.2.1 Sistema de Controle Supervisório

A Figura 33 ilustra a divisão de um sistema de controle supervisório dentro da pirâmide da automação. Esta divisão possibilita sistematizar a implementação do sistema de controle supervisório.

No nível 1 estão os equipamentos de campo, como atuadores e sensores, além dos botões de acionamento local. No nível 2 e 3 estão o CLP e o sistema SCADA respectivamente, responsáveis pelo controle dos equipamentos do nível 1. No nível 4 está o MES, responsável pela execução das tarefas e onde é implementado o SRT. E na última camada está o PCP.

4.2.2 Arquitetura de Sistema do Controle Supervisório Integrado

Na Figura 5 é apresentada a arquitetura de implementação do controle supervisório em três níveis hierárquicos (QUEIROZ; CURY, 2002b).

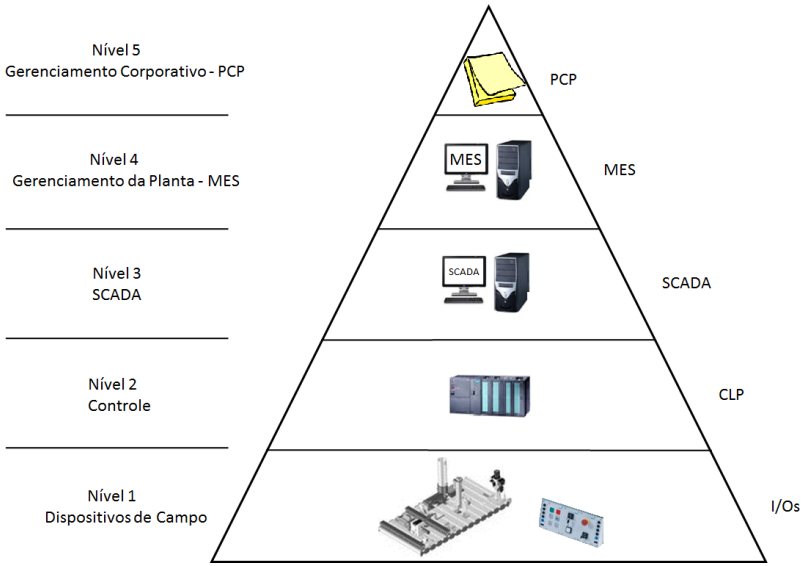

Figura 33 – Divisão do Sistema de Controle Supervisório.

Nesse trabalho, é desenvolvida uma extensão dessa arquitetura com as etapas de implementação do SCADA e do SRT. A nova arquitetura é dividida em três *hardwares*, que juntos compõem o Sistema de Controle Supervisório. Além do *hardware* de coordenação, que é o código implementado no CLP, a nova arquitetura conta com um *hardware* referente a implementação no SCADA e outro referente à implementação do SRT. A Figura 34 mostra a nova arquitetura dividida em três blocos (SRT, SCADA e CLP).

Na arquitetura original, o *hardware* de coordenação tem três níveis hierárquicos, que são implementados no CLP: Supervisores Modulares; Sistema Produto; e Sequências Operacionais. Neste trabalho é adicionado um quarto nível, o qual trata da implementação do Gerenciador de Modos de Operação (GMO) (ver Figura 26). Esse nível define o modo de operação a ser executado pelo sistema produto. Caso seja executada alguma ação que leve a um estado de bloqueio da lógica, o GMO recebe do sistema produto a informação de lógica não segura.

O *hardware* SCADA tem apenas um nível, o qual troca informações sobre os modos de operação com o GMO, recebe informações sobre o estado atual do sistema produto para realizar as representações gráficas, recebe dos supervisores as informações de quais eventos estão

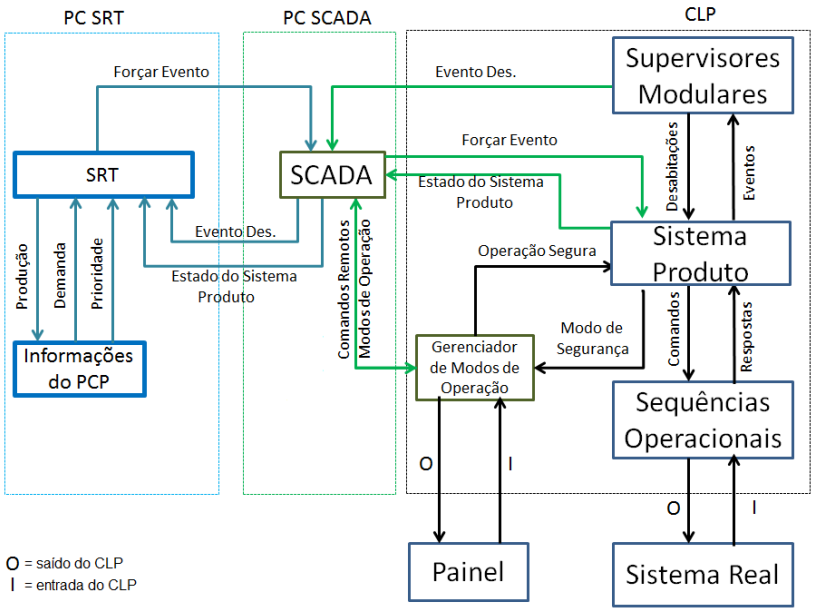

Figura 34 – Arquitetura de Sistema do Controle Supervisório Integrado.

desabilitados e permite ao operador habilita os eventos quando em modo manual.

O terceiro *hardware* é o SRT, o qual quando habilitado pelo SCADA, recebe informações do mesmo sobre os eventos que estão desabilitados e informações sobre os estados atuais de cada sistema produto, além de receber do PCP as informações para realizar o roteamento das tarefas.

4.2.3 Gerenciador de Modos de Operação

Na Figura 26 é apresentado o GMO, desenvolvido por Vieira (2007). Para esse projeto, foi necessário realizar adaptações no GMO, devido ao novo modo de operação "SRT", responsável pelo roteamento das tarefas. As adaptações podem ser visualizadas na Figura 35.

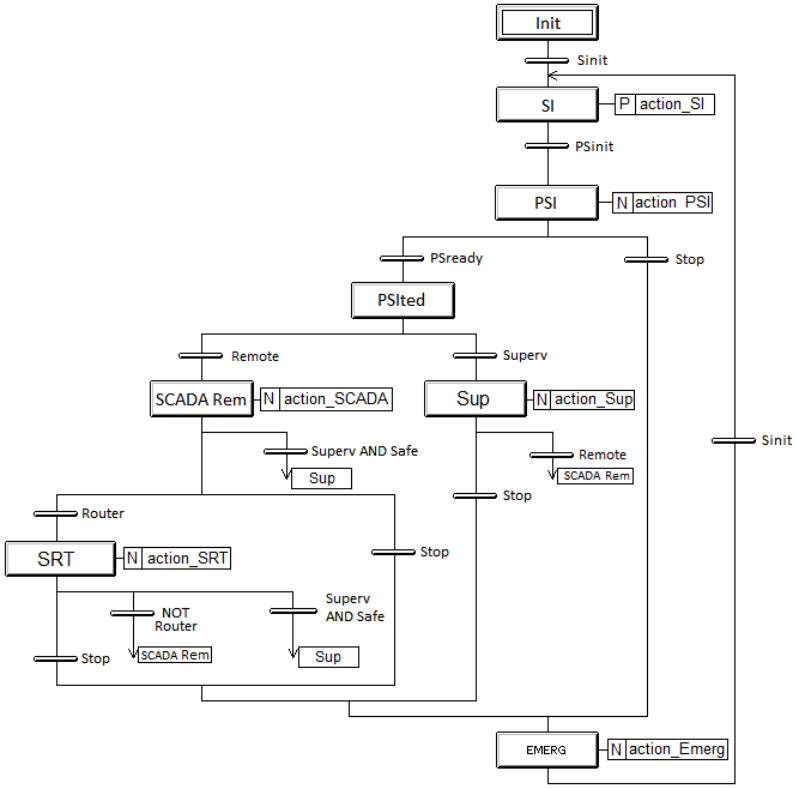

Figura 35 – Gerenciador de Modos de Operação.
Fonte: Adaptado de (VIEIRA, 2007).

Primeiramente foi definido que o modo de operação "SRT"só pode ser ativado se a lógica não estiver executando de forma automática (modo Sup), por isso para ativar o SRT o sistema deve estar em modo "SCADA Remoto". Estando nesse modo, a condição para ativar o modo SRT é clicar no botão "Ativar SRT"criado na tela do sinótico. Se o sistema estiver no modo "Sup"esse botão fica desabilitado.

No modo "SRT"a execução dos comandos é através do SRT. Para voltar para o modo "SCADA Remoto"basta clicar novamente no botão "Ativar SRT", o que resulta na desativação do modo. Também é possível passar do modo "SRT"diretamente para o modo "Sup", bastando girar a chave na IHM para a posição "Superv"e ter a condição de lógica "Safe"(Operação segura, ou seja, nenhum evento que cause

bloqueio foi executado, permanecendo dentro da lógica de controle supervisório).

No modo "SCADA Remoto"(na Figura 26 esse modo é chamado de "MAN") o GMO permite a mudança para o modo "Sup", desde que a lógica tenha condição "Safe"e a chave da IHM mude para a posição "Superv".

As demais condições do GMO ocorrem conforme já explicado na seção 3.3.4.

4.3 METODOLOGIA DE DESENVOLVIMENTO DE SISTEMA DE CONTROLE SUPERVISÓRIO INTEGRADO AO ROTEAMENTO DE TAREFA

A proposta desse trabalho é estender a metodologia desenvolvida por Portilla (2011) com uma fase que trate da implementação do SRT. Essa fase deve solucionar o problema causado pelo grau de liberdade resultante da TCS. A nova metodologia tem um total de nove fases. A fase chamada de "Implementação do SRT"é adicionada como oitava etapa da metodologia, como é visto na Figura 36.

A fase "Implementação do SRT"é dividida em duas partes, que são:

- Instalação e configuração do SRT: É realizada a instalação do *Software* que será responsável pelo roteamento. Também é definido o protocolo de comunicação entre o programa que realiza o roteamento e o sistema SCADA. Por fim, é realizado um teste de troca de dados entre os programas;

- Implementação da Lógica de Roteamento: É definida uma heurística para a escolha dos eventos. Depois é desenvolvido o algoritmo com base nessa heurística para realizar o roteamento das tarefas. Por fim são realizados pequenos testes para verificar o funcionamento da lógica de roteamento.

A Fase IV da metodologia é alterada com objetivo de separar um passo específico para implementar o GMO, visto que a metodologia de Portilla (2011) não prevê essa implementação. Foi definida a etapa 4.2 para o tratamento do GMO.

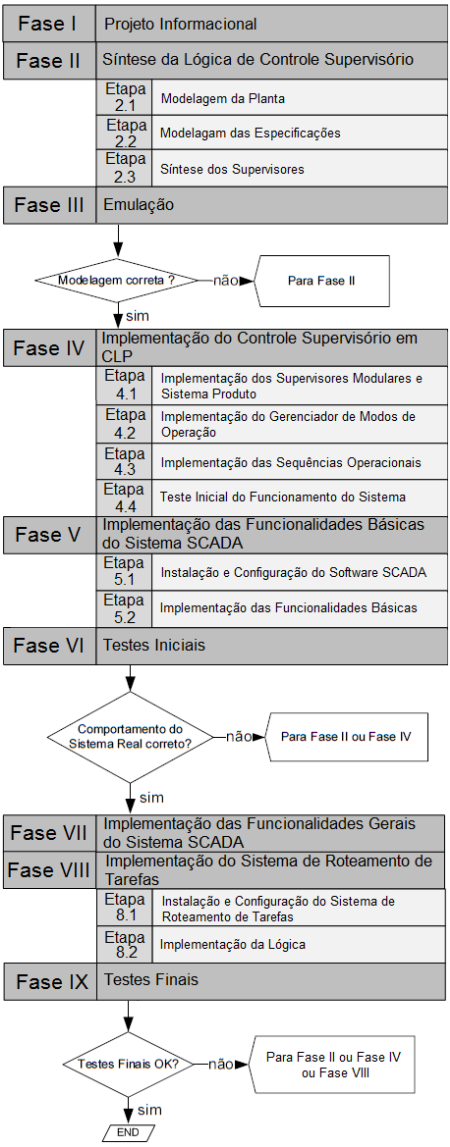

Figura 36 – Extensão da Metodologia de Integração.
Fonte: Adaptado de (PORTILLA, 2011).

4.4 APLICAÇÃO DA METODOLOGIA AO MÓDULO DISTRIBUIÇÃO

A seguir, é apresentado o desenvolvimento da metodologia aplicada na Estação de Distribuição da bancada da Festo, a mesma apresentada na Seção 3, mas agora seguindo a metodologia apresentada na Figura 36.

4.4.1 Projeto Informacional, Síntese de Controle Supervisório e Emulação

No projeto informacional é mantido o que já foi desenvolvido, mas com o complemento da descrição das ferramentas necessárias para o desenvolvimento da fase 8. As ferramentas utilizadas são: *Matlab* para implementar o algoritmo de roteamento das tarefas; e o Simatic Net (OPC Scout V10) para realizar a comunicação *OLE for Process Control* (OPC) entre o SCADA e o *Matlab*.

Para o desenvolvimento do SRT é necessário uma ferramenta que execute um algoritmo de controle, além de comunicar-se com o SCADA, para troca de informações. A ferramenta escolhida para esse projeto é o *Matlab* (GUIDE, 1998), o qual realiza comunicação com o SCADA via OPC. A escolha se deve ao fato de ser uma ferramenta que o LAI tem a disposição para estudos e por ser uma ferramenta de conhecimento do autor deste trabalho.

O *Matlab* é o sistema responsável pelo roteamento das tarefas. Ele analisa as informações do controle supervisório para saber quais eventos estão desabilitados no CLP em um determinado ciclo. Com esses dados, o *Matlab* roda o algoritmo a fim de escolher o evento que deve ser executado. Informações providas do PCP podem definir qual evento será executado.

A Síntese de Controle Supervisório e a Emulação se mantiveram da mesma forma como desenvolvido na Seção 3, já que o funcionamento da planta é o mesmo.

4.4.2 Implementação do Controle Supervisório em CLP

A implementação da modelagem do controle supervisório em CLP é realizada na primeira etapa desta fase. Como a lógica de controle supervisório não teve alteração em relação ao desenvolvido no Capítulo 3, essa etapa manteve-se inalterada.

Na segunda etapa da Fase IV é aplicado o GMO. Essa etapa é importante para a continuidade da metodologia, ante a necessidade de ter uma inicialização do sistema, de ter uma definição do modo de operação e de ter um modo que trate das situações de emergência já durante os testes iniciais.

Implementado o GMO, pode-se passar para a terceira etapa desta fase, a qual implementa as sequências operacionais. Esta etapa não teve alterações, já que o modo "SRT"e o modo "SCADA Remoto"utilizam as mesmas variáveis para executar os comandos. Ambas utilizam a variável $force\_ < evento >$, a única diferença é que esta variável ou é setada na tela do sinótico quando em modo "SCADA Remoto", ou é setada pelo SRT quando em modo "SRT".

Por fim, são realizados os testes iniciais do funcionamento do sistema. Nenhum erro foi encontrado durante os testes, isso se deve às poucas mudanças no desenvolvimento em relação à metodologia do Capítulo 3.

4.4.3 Implementação das Funcionalidades Básicas e Gerais do Sistema SCADA e Testes Iniciais

Nas fases V, VI e VII da metodologia praticamente não houve mudanças em relação ao que já fora desenvolvido no Capítulo 3. A única modificação é a implementação de um botão para acionamento do modo "SRT"na tela do sinótico, o qual habilita o SRT para gerenciar a escolha dos eventos. Outras funcionalidades devem ser implementadas, como um campo para mostrar o número de peças a serem produzidas ou o número de peças que faltam para finalizar um pedido, mas essas informações devem vir do PCP e como essa etapa ainda não foi desenvolvida, essas implementações serão adicionadas posteriormente.

4.4.4 Implementação do Sistema de Roteamento de Tarefas - SRT

Na oitava fase da metodologia é desenvolvido o SRT. Esse sistema é implementado no programa *Matlab*, o qual comunica-se com o SCADA e CLP via OPC, além de acessar informações do PCP através de uma planilha de dados. A escolha dos eventos segue métodos de roteamento heurísticos fundamentados em prioridades de eventos e demanda de produção.

As seguintes etapas compõem a fase de implementação do SRT: (*i*) Instalação e Configuração do SRT; e (*ii*) Implementação da heurística de Roteamento.

(i) - Instalação e Configuração do SRT.

A primeira etapa dessa fase é separada para a instalação e configuração do *software*. O *Matlab* é instalado em uma máquina com sistema operacional *Windows* do LAI-UFSC. A comunicação entre o *Matlab* e o SCADA acontece via servidor OPC. O *software* utilizado para realizar a comunicação é o OPC Scout V10, da própria *Siemens*, o qual usa uma estrutura Cliente/servidor, onde ele será o servidor e o Matlab e o SCADA serão os clientes. O nome do servidor gerado é *OPC.SimaticHMI.CoRtHmiRTm*. Para que o *Matlab* possa se comunicar com o OPC é necessário ter instalado o pacote *OPC Toolbox*. A versão do *Toolbox* utilizada é 3.1.2 (R2012b).

A Figura 37 ilustra os códigos utilizados para realizar a comunicação entre SCADA e *Matlab* e para ler as *Tags* utilizadas para forçar a ocorrência dos eventos através do CLP. O nome da conexão realizada com o servidor OPC é "da"e "grp"é o nome do grupo de variáveis que estão relacionadas a esse servidor. Um teste inicial de leitura e de escrita de alguma variável é realizado para verificar a comunicação entre SCADA e *Matlab*.

```
% % ----- REALIZA CONEXAO
da = opcda('localhost','OPC.SimaticHMI.CoRtHmiRTm');
connect(da);
grp = addgroup(da);
set(grp, 'UpdateRate', .500);

% ----- ESPECIFICA TAGS QUE SERAO LIDAS/ESCRITAS
itm0 = additem(grp, 'force_aliment_avanca');
itm1 = additem(grp, 'force_aliment_retorna');
itm2 = additem(grp, 'force_braco_avanca');
itm3 = additem(grp, 'force_braco_recua');
itm4 = additem(grp, 'force_vent_ejeta');
itm5 = additem(grp, 'force_vent_pega');
```

Figura 37 – Código utilizado para realizar a comunicação entre *Matlab* e *WinCC*.

(ii) - Implementação da lógica de Roteamento.

A implementação da lógica de roteamento leva em conta a leitura das informações da lógica de controle supervisório implementadas

no CLP. Essas informações visam definir os estados atuais dos sistemas produtos (para saber quais são os eventos fisicamente possíveis de ocorrer) e estabelecer quais os eventos que estão desabilitados pelos supervisores. Com essas informações é implementado uma heurística de roteamento, a qual deve forçar a ocorrência dos eventos no CLP.

Como a lógica de controle supervisório trabalha com uma atualização a cada ciclo da CPU, a lógica do *Matlab* não deve executar mais de um evento por ciclo da CPU. O que não ocorre normalmente, visto que a execução do código no *Matlab* é mais rápida.

Para resolver o problema foi utilizada a variável interna do CLP *LOGCODE*, gerada pelo IDES2ST. Em cada estado do sistema é atribuído um valor diferente para a variável *LOGCODE*, o que faz com que o valor dessa variável seja diferente em todos os estados dos sistemas produtos. O *Matlab* compara o valor da *LOGCODE* com uma variável interna do Matlab chamada *LOG*, caso esse valor seja diferente, ele executa a lógica e muda o valor da *LOG* para o mesmo da *LOGCODE*. Desta forma, tem-se a garantia que a lógica só é executada quando os valores de *LOGCODE* e *LOG* forem divergentes, ou seja, o sistema jamais executa dois eventos antes de uma atualização no sistema produto.

4.4.5 Testes Finais

Após todas as etapas da Metodologia serem implementadas, são realizados os Testes Finais, onde a planta é submetida a inúmeros testes para verificar o correto funcionamento. Caso algum problema seja encontrado, é necessário identificar a causa e retornar para a fase II, IV ou VIII, conforme necessário, para que então sejam realizadas as mudanças adequadas. Esta iteração é realizada quantas vezes necessárias, até que o sistema se comporte de forma satisfatória.

Como boa parte da implementação já havia sido testada na implementação apresentada no Capítulo 3, poucos ajustes foram necessários. O principal problema encontrado foi em relação ao tempo de execução do *Matlab* em relação ao do CLP, causando a execução de mais de um evento antes que o controle do CLP fosse atualizado. Após o retorno para a fase VIII, foi corrigido o problema e realizados os testes finais novamente. Então, finalmente o SRT executou de forma satisfatória, conforme o proposto para esse trabalho.

4.5 HEURÍSTICAS DE ROTEAMENTO

Uma característica do MPS é que a lógica de controle é genérica, ou seja, a lógica de controle da planta resolve apenas questões de segurança e sequenciamento de cada módulo, e não depende da produção. O MPS trabalha com vários módulos e cada um deles tem uma lógica de controle individual, que é independente da produção e de outros módulos. Para que todos os módulos trabalhem em conjunto, um controle de sequência de produção é necessário. No caso desse trabalho, a definição da sequências de produção é definida pelo SRT.

Nesta seção é apresentada a heurística de implementação no SRT para a escolha dos eventos.

4.5.1 Prioridade Definida pelo PCP

O SRT é um sistema genérico que permite a implementação de uma determinada heurísticas para realizar a escolha de eventos. A heurística utilizada nesse trabalho baseia-se na prioridade de eventos e demanda de produção. Todos os eventos são definidos com um grau de prioridade segundo informações do PCP. Uma característica dessa heurística é o controle de produção por demanda, onde o SRT só produz o número predefinido de peças.

A Figura 38 mostra a arquitetura de implementação do SRT.

No primeiro passo é realizada a leitura de informações do CLP. O segundo passo deve ler as informações contidas na planilha. No terceiro passo é realizada a execução do algoritmo programado no SRT. Na quarta e última etapa é executado o evento, onde o SRT manda a informação do evento escolhido para o CLP, o qual força o evento e atualiza a lógica de controle supervisório.

A definição da prioridade leva em conta o *layout* dos módulos do MPS e o arranjo dos eventos de acordo com o fluxo de produção. Neste trabalho, a prioridade maior é definida para os eventos mais perto do final do processo, a fim de terminar primeiro a tarefa em execução mais rápido quanto possível e, portanto, minimizar o tempo de ciclo. De forma geral, quando houver um conflito de dois eventos possíveis de ocorrer, o SRT deve dar prioridade ao evento que está mais próximo do final do processo. Uma opção seria inverter a definição de prioridade, ou seja, os eventos no início do processo terem maior prioridade, o que resulta em maximizar o trabalho em andamento.

O SRT lê informações de uma planilha de dados fornecida pelo

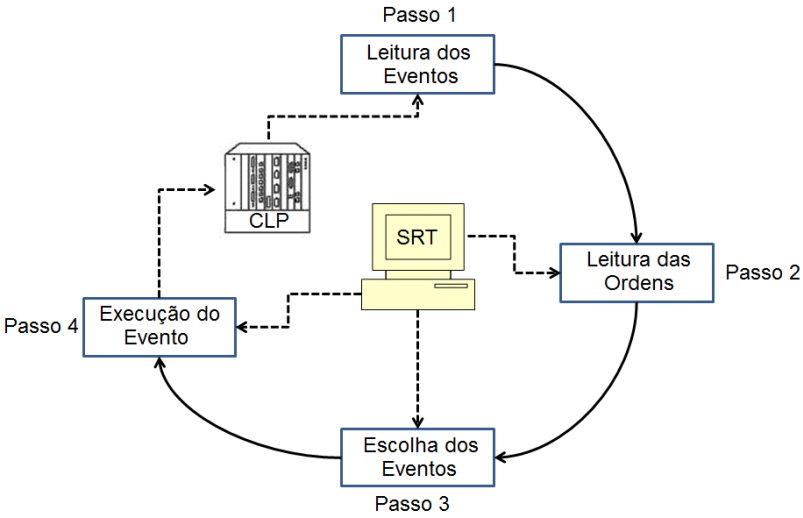

Figura 38 – Arquitetura de Implementação do SRT.

PCP, a fim de escolher os eventos a serem executados. Essa planilha contém informações de prioridade dos eventos (eventos controláveis e não controláveis), de demanda de produção, do número de peças produzidas e do número de peças que estão em produção.

Nessa heurística também são utilizados os eventos não controláveis como opção de escolha. Caso o evento com maior prioridade seja um evento não controlável, o SRT retorna para o operador "Esperando o evento não controlável 'nome do evento' terminar de executar", nesse caso nenhum evento é executado.

Na tela do sinótico são implementados campos com informações para visualização do operador, a fim de indicar o número de peças que devem ser produzidas por lote, o número de peças que estão em produção e o número de peças que já foram produzidas. Caso o número de peças produzidas atinja a demanda requisitada, é indicado um alerta para o operador.

O código implementado para essa heurística pode ser visto no apêndice A, o qual funciona da seguinte forma: o SRT inicia verificando quais eventos estão elegíveis; depois o SRT carrega os dados da planilha disponibilizada pelo PCP; o SRT verifica a demanda de produção, quantas peças estão em produção e quantas peças já foram produzidas, e escreve esses dados no SCADA para visualização; o SRT verifica qual

evento tem maior prioridade dentre aqueles que estão habilitados; caso tenha mais do que um evento com o mesmo grau de prioridade, é utilizado o algoritmo randômico para escolha de um deles (esse algoritmo realiza o sorteio de um número que corresponde a um evento); o SRT força a execução do evento escolhido e muda o valor da variável *LOG* para o mesmo da variável *LOGCODE*; caso tenha apenas um evento habilitado, o SRT vai executá-lo; caso o evento escolhido seja um evento não controlável, o SRT não executa nenhum e mostra uma mensagem para o operador; os dados de peças em produção e de peças produzidas são atualizados na Planilha e no SCADA; o Algoritmo aguarda até ser atualizada a lógica no CLP para reiniciar o ciclo; se o valor da demanda for igual ao valor de peças produzidas, um alerta no SCADA é acionado; se o número de peças produzidas somado ao número de peças em produção for igual ou maior do que a demanda, o SRT não deixa o alimentador colocar mais nenhuma peça para produção, bloqueando o evento *Avanca_alimentador*.

O algoritmo randômico é implementado nessa heurística devido ao fato de que se um evento deve ser executado antes que outro, ele deve ter uma prioridade maior, mas se ele tem a mesma prioridade, qualquer um que executar não influenciará no planejamento da produção.

Por fim, foram realizados testes com a implementação do SRT utilizando informações do PCP para definição de prioridade de escolha de eventos. Diversos testes foram realizados, tanto na escolha de eventos por parte do SRT quanto no controle de demanda de produção e na mudança do grau de prioridade durante a execução. O único problema encontrado foi em relação à mudança de prioridade de um evento durante a execução. O problema só aparecia quando a mudança de prioridade na planilha ocorria no exato momento em que o *Matlab* tentava acessar aquela informação. Esse problema era evitado fazendo a mudança da prioridade no momento certo.

5 APLICAÇÃO DA METODOLOGIA NO MÓDULO ESTAÇÃO DE TESTE

Nos capítulo anterior foi apresentada a metodologia para desenvolvimento de sistema de controle supervisório integrado ao CLP, ao SCADA e ao SRT. Essa metodologia foi aplicada ao módulo Distribuição da bancada da Festo. A metodologia foi testada e validada, obtendo ótimos resultados, principalmente em relação ao desenvolvimento do SRT.

Neste capítulo é aplicada toda a metodologia desenvolvida a um segundo módulo da bancada, o módulo Estação de Teste. Essa implementação serve pra avaliar todo o desenvolvimento do Capítulo 4 e mostrar a eficácia da teoria aplicada em sistemas modulares de produção, além da fácil adaptação na utilização da metodologia em diversos módulos em conjunto.

A seguir, são apresentados de forma simplificada todos os passos do desenvolvimento do módulo Estação de Teste, utilizando a metodologia apresentada no Capítulo 4.

5.1 PROJETO INFORMACIONAL

A Estação de Teste tem como objetivo analisar as peças que chegam, verificando a espessura das mesmas. Caso o tamanho da peça esteja irregular, a mesma é reprovada e expulsada na esteira de baixo, mas se a peça estiver com a espessura correta ela é encaminhada pela esteira superior para o próximo módulo. A Figura 39 mostra o módulo Estação de Teste.

O módulo Estação de Testes é composto por:

- 1 sensor capacitivo para detecção de chegada de peças;
- 1 sensor reflexivo para monitoramento do espaço de trabalho do cilindro de elevação;
- 2 sensores de proximidade acoplados ao pistão do cilindro de elevação para detecção das posições finais do cilindro;
- 1 sensor comparador que serve para verificar a espessura da peça;
- 1 sensor Barreira Luminosa (IP_FI) que está relacionado com o bloqueio/liberação do módulo seguinte;

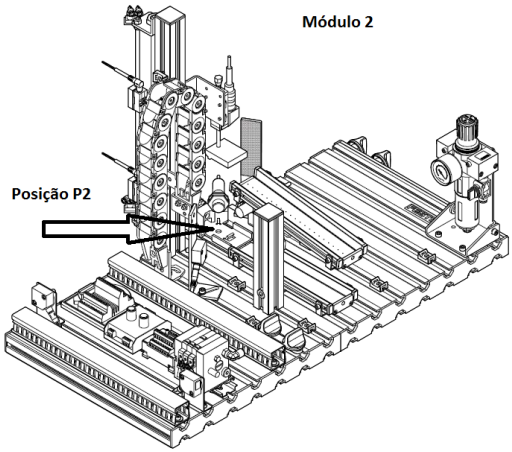

Figura 39 – Bancada de Teste MPS-Festo.
Fonte: (EBEL; PANY, 2006).

- 1 sensor Barreira Luminosa (IP_FO) que está relacionado com o seu bloqueio/liberação, esta informação é enviada ao módulo anterior;
- 2 sensores de proximidade acoplados ao cilindro de ejeção de peças para indicar as posições finais do cilindro;
- 1 cilindro de elevação (elevador);
- 1 cilindro de ejeção de peças (ejetor);
- 1 CLP Siemens S7-1200;
- 1 painel IHM;

As demais informações, como também os softwares utilizados e protocolos de comunicação foram os mesmos utilizados na Seção 3.3.1.

5.2 SÍNTESE DE CONTROLE SUPERVISÓRIO E EMULAÇÃO

A metodologia segue a abordagem modular local. Para um correto funcionamento do módulo Estação de Teste, é necessário definir as especificações para restringir o comportamento da planta e evitar problemas. Deste modo, as seguintes especificações são definidas:

1. Caso o espaço de trabalho esteja ocupado, o elevador e o ejetor não devem acionar;

2. Exclusão mútua entre a subida do elevador e o avanço do ejetor;

3. Exclusão mútua entre a descida do elevador e o avanço do ejetor;

4. O elevador não deve subir sem que haja uma peça na posição P2 (posição em que chegam as peças)

5. O ejetor não ejeta peças caso a módulo seguinte esteja ocupado (informação indicada pelo sensor IP_FI). Mas isso só deve restringir a ejeção quando o elevador estiver elevado, caso ele esteja abaixado a ejeção é independente do estado do módulo seguinte;

6. O ejetor só pode acionar depois da indicação de que o módulo está ocupado;

7. O elevador não deve descer sem antes ejetar as peças aprovadas;

8. O elevador só pode acionar depois da indicação de que o módulo está ocupado;

9. O módulo deve indicar ao módulo anterior (por meio do sensor IP_FO) qual o seu estado (liberado quando sem peça ou ocupado quando chega uma peça).

Definidas as restrições do sistema, é necessário modelar os subsistemas que integram o módulo Estação de Testes. O módulo é dividido em seis subsistemas, onde cada um é representado por um autômato. Os subsistemas são: (1) Indicação de espaço de trabalho ocupado; (2) Indicação que o módulo seguinte está ocupado; (3) Indicação que o módulo Estação de Teste está ocupado; (4) Ejetor de peças; (5) Elevador; (6) Indicação de chegada de peças. A Figura 40 mostra a modelagem de todos os subsistemas.

O próximo passo é realizar a modelagem das especificações, conforme as restrições já definidas. Para cada item das restrições é modelado um autômato. A Figura 41 mostra a modelagem das 9 especificações.

Então foi aplicada a TCS seguindo a abordagem modular local, para obter os supervisores locais. O resultado foram 9 supervisores, os quais foram sincronizados para verificar a ação conjunta e a modularidade entre eles. Obteve-se um supervisor geral de 408 estados não-bloqueantes.

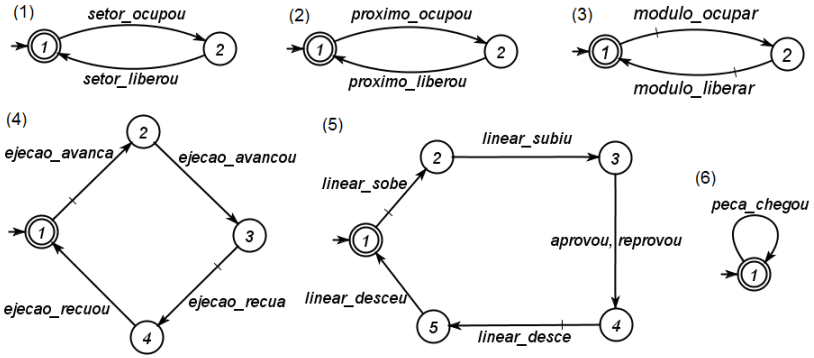

Figura 40 – Sistema Produto

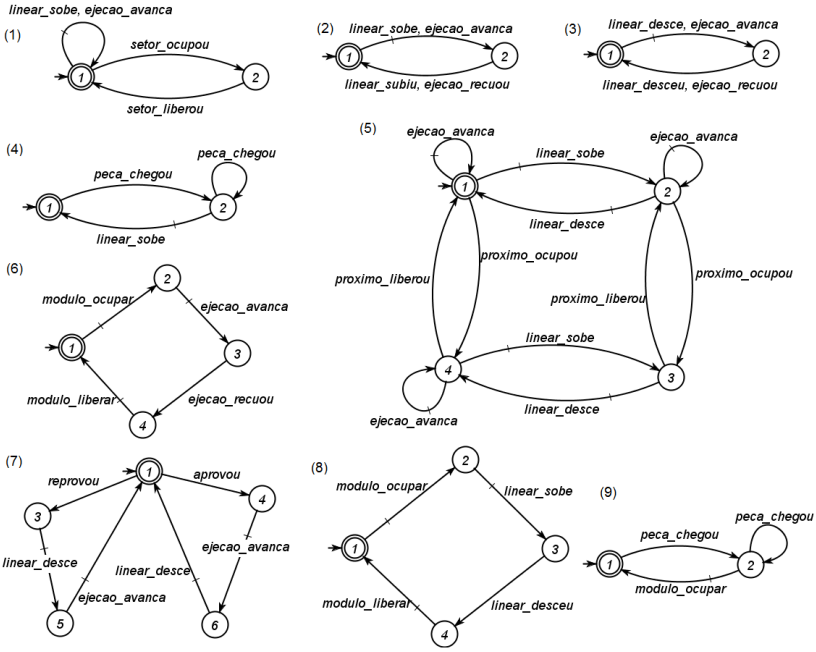

Figura 41 – Especificações

Em seguida, foram encontrados os supervisores reduzidos com o objetivo de diminuir o uso de memória computacional e facilitar a interpretação dos supervisores durante a implementação no CLP.

Por fim, foram realizados testes de funcionamento da lógica por meio da ferramenta Supremica. Alguns ajustes na lógica foram necessários para que o comportamento do controle supervisório estivesse de acordo com as restrições definidas.

5.3 IMPLEMENTAÇÃO DO CONTROLE SUPERVISÓRIO EM CLP E DESENVOLVIMENTO DO SISTEMA SCADA

As fases 4, 5, 6 e 7 seguem as mesmas configurações e ajustes realizados na Seção 3.2. Nenhuma nova modificação na lógica, além daquelas realizadas para o módulo Distribuição, foram necessárias.

As telas do sinótico foram desenvolvidas com base na planta, e diferente do módulo Distribuição, não foi necessário utilizar valores dos supervisores para indicações gráficas, apenas sendo necessário os valores dos sistemas produtos. Isso se deve ao fato de todas as posições necessárias terem sensores para indicação, diferente da Posição P1 do módulo Distribuição.

Os Testes Inicias mostraram o bom sincronismo dos dois módulos modelados com a TCS. Alguns problemas foram encontrados, principalmente em relação ao sincronismo de controle de aprovar e reprovar peças. A especificação 7 resolveu esse problema. Quando o sistema se comportou sem nenhum bloqueio e de acordo com o desejado, foi iniciada a Fase VIII.

5.4 IMPLEMENTAÇÃO DO SISTEMA DE ROTEAMENTO DE TAREFAS E TESTES FINAIS

Nesta fase foi desenvolvido o SRT para o módulo Estação de Testes, possibilitando assim o controle de produção por meio de definições do PCP. Como já havia sido desenvolvida toda a lógica e estabelecida uma heurística de roteamento das tarefas para o primeiro módulo, foi necessário apenas adicionar os novos eventos na lógica e fazer algumas adaptações, para então permitir ao SRT o controle dos dois módulos do MPS.

Vários testes e diversos ciclos de produção de peças foram realizados, alguns problemas foram encontrados e corrigidos até chegar na lógica desejada, livre de bloqueio e com o funcionamento seguindo dentro da lógica de controle supervisório estabelecida.

No final foi possível realizar uma simples comparação da lógica

desenvolvida utilizando a Metodologia de Desenvolvimento de Sistema de Controle Supervisório integrando CLP, SCADA e um Roteamento de Tarefas, com a lógica original da bancada desenvolvida pelo fabricante. A nova lógica possibilita uma maior segurança, devido à implementação de alarmes, relatórios de eventos e tratamento de situações de parada por emergência. Também obteve-se um maior número de funcionalidades (disponibilidade de três modos de trabalho, evolução passo-a-passo e acionamentos remotos), além de uma maior dinâmica no controle do sistema.

Nos testes realizados, constatou-se que na utilização das duas bancadas (Distribuição e Estação de Teste), o SRT se mostrou mais lento na escolha dos eventos, devido a uma quantidade maior de eventos. Esse problema não resultou em atrasos de produção, mas talvez em grande escala possa resultar em atrasos significativos.

Depois de todos os testes o sistema é validado, sendo aprovada a utilização de um SRT para escolha dos eventos, com informações providas do PCP. A utilização de dois módulos mostrou-se flexível e de fácil adaptação do sistema às diversas mudanças nas rotas de produção.

6 CONCLUSÃO

Este trabalho apresentou uma metodologia baseada na TCS para sintetizar e implementar um sistema de controle supervisório que integra CLP, SCADA e Roteamento de Tarefas. A aplicação em um MPS didático mostrou que a síntese formal e emulação de supervisores modulares reduzidos, minimamente restritivos e não bloqueantes levou ao desenvolvimento de uma lógica de controle que é flexível, segura e compreensível. Além disso, a arquitetura de controle proposta ajuda o sistema de roteamento de tarefas, combinado com a eficiência de métodos de planejamento de produção a ter um sistema de controle de supervisão implementado em CLP com alta flexibilidade de reconfiguração e segurança.

Como principais contribuições desse trabalho, podemos citar o desenvolvimento de um sistema que faz o roteamento das tarefas, afim de possibilitar a escolha dos eventos, determinando assim a rota a ser seguida. Podemos citar também a consolidação da metodologia para desenvolvimento de um sistema de controle supervisório para um Sistema Modular de Produção, integrando o CLP, o SCADA e o SRT. Além de uma heurística desenvolvida para realizar o roteamento de tarefas, de uma arquitetura de sistema de controle supervisório integrado e da adaptação do Gerenciador de Modos de Operação ao o SRT.

A escolha pelo desenvolvimento em um MPS, foi motivada pelos estudos crescentes dessas estruturas modulares. Isso é fruto dos novos objetivos econômicos na atualidade, que veem a necessidade de adaptar com agilidade e baixo custo os sistemas de produção em relação as condições do mercado. E é nesse cenário que se encaixa o roteamento de tarefas, objetivo principal desse trabalho, o qual possibilita a configuração de diversas rotas de produtos sem realizar uma nova modelagem de lógica, sendo necessário apenas a mudança de prioridade dos eventos.

A metodologia de integração de sistemas SCADA com implementação de controle supervisório em CLP já havia sido validada por Portilla (2011) em um Sistema Flexível de Manufatura (FMS). Esse trabalho aplicou a metodologia ao módulo Distribuição e ao módulo Estação de Teste que fazem parte da bancada MPS didática da Festo. A metodologia foi estendida com a implementação do SRT para a escolha dos eventos, além de adicionar uma etapa específica para implementação de um Gerenciador de Modos de Operação.

Foi implementada a heurística de Prioridade definida pelo PCP,

onde o sistema executa as ações conforme a lógica de controle supervisório, mas levando em consideração o grau de prioridade dos eventos, principalmente para situações em que mais de um evento controlável estiver elegível. Nessas situações, quem define a ação a ser executada é o SRT, que força a ocorrência do evento conforme informações vindas do PCP. Em geral, a definição de uma prioridade mais alta para um evento em relação a outro define o caminho a ser percorrido.

Utilizando técnicas tradicionais de programação, onde a mesma é realizada de forma sequencial e direta para um tipo de produto, é necessário realizar toda uma reprogramação da lógica quando existir a necessidade de mudar a rota de um produto ou mudar o *layout* da planta. Com a utilização de um SRT isso não ocorre, em muitos casos essa reprogramação pode não ser necessária, bastando apenas mudar os graus de prioridades dos eventos para que a rota seja alterada.

Um exemplo em que o SRT poderia ser utilizado e traria benefícios, é encontrado no problema da modelagem do controle supervisório, citado na Seção 3.3.6 e ilustrado na Figura 31. Neste caso, havia uma situação em que dois eventos estavam habilitados e a execução de um deles levava a um problema. Isso poderia ser facilmente resolvido dando prioridade para o evento que não levava ao estado que causava bloqueio. Para resolver o problema, foi necessário criar uma nova especificação, visto que o estudo de implementação do SRT ainda estava em desenvolvimento e o problema tinha que ser resolvido naquele momento.

A aplicação da metodologia também traz vantagens em relação à TCS utilizando a abordagem modular local, já que utiliza técnicas formais para a modelagem, prevenindo erros de programação e erros de modelagem de plantas e especificações já nas primeiras fases do projeto. Quando encontrado algum problema que não foi identificado inicialmente, é possível desenvolver uma nova especificação e gerar um novo supervisor modular local para corrigir o problema. Esse supervisor modular local não altera o que já havia sido implementado, apenas adiciona um novo supervisor na lógica de controle supervisório.

Ainda sobre a modelagem, outro ponto importante é a escolha das ferramentas adequadas para o desenvolvimento. A ferramenta Supremica foi de suma importância para o desenvolvimento e testes da lógica de controle supervisório. Com a ferramenta era possível realizar testes de simulação facilmente, desenvolver os supervisores e verificar a presença de bloqueios na lógica. Outra ferramenta importante e que diminuiu consideravelmente o tempo de implementação da lógica de controle supervisório no CLP foi o IDES2ST, a qual gerou o código em

linguagem Texto Estruturado para implementar no CLP automaticamente, sendo necessário alguns ajustes devido as novas funcionalidades implementadas.

É importante destacar a integração que foi realizada entre todos os sistemas envolvidos (CLP, SCADA e SRT) seguindo a arquitetura de implementação desenvolvida. O CLP controla todo o processo, o SCADA permite uma visualização gráfica da planta, de alarmes e de relatórios, além de acionamentos remotos, e o SRT define o roteamento das tarefas e controla a demanda de produção.

Na literatura encontramos apenas trabalhos que tratam da escolha dos eventos de forma aleatória ou de alternância entre as escolhas, ou seja, sem um controle de qual evento deve executar. O único trabalho na literatura encontrado que trata da escolha dos eventos com base em critérios de prioridade é o trabalho de Molina (2007), mas o mesmo utiliza o formalismo de Redes de Petri para tratar essa escolha de eventos. A principal dificuldade para o desenvolvimento do SRT foi o encontro de poucos trabalhos correlacionados.

Os testes realizados com a implementação do SRT nos módulos Distribuição e Estação de Teste da bancada didática da Festo, mostram a eficiência de métodos de planejamento de produção em relação à segurança e flexibilidade do controle supervisório em CLP. Outro resultado positivo é em relação a aplicação da lógica de controle supervisório utilizando um sistema SCADA, onde é possível a utilização de valores dos sistemas produtos para visualização gráfica da planta,

Esse trabalho deixou em aberto alguns pontos para aperfeiçoar o SRT, que podem ser explorados em trabalhos futuros. Um dos pontos é o desenvolvimento de novas heurísticas, com o objetivo de diminuir o tempo de interação entre o SRT e o controle supervisório implementado no CLP. Conforme apresentado na Seção 5.4, quanto mais bancadas é utilizado, maior é o número de eventos utilizados e mais lento fica o SRT. Outro ponto a ser explorado em outras heurísticas é a forma de mudança das prioridades, implementando uma forma de mudar a prioridade sem causar problemas. Podemos citar ainda o estudo de viabilidade de aplicação do SRT em um MES comercial, que se integre diretamente com o PCP.

Uma geração automática de código também é um campo de interesse para estudos futuros, visto que a utilização do IDES2ST em um primeiro momento foi satisfatório, mas ao decorrer do trabalho, várias mudanças foram necessárias. Então um programa que gerasse o código mais completo, visando a implementação do SRT, traria ganhos de tempo durante o desenvolvimento da metodologia.

Por fim, podemos citar o estudo para implementar a metodologia em todos os setes módulos da bancada Festo, o que aperfeiçoaria a metodologia, tendo em vista que o desenvolvimento teria que lidar com um maior número de eventos, uma maior possibilidade de rotas alternativas de produção, além de lidar com novos cenários.

REFERÊNCIAS

AKESSON, K. *Methods and Tools in Supervisory Control Theory*. Tese (Doutorado) — Chalmers University of Technology, Sweden, 2002.

ASSOCIATION, M. E. S. M e s explained: A high level vision. In: *MESA International - White Paper Number 6*. [S.l.: s.n.], 1997.

BO, L.; ZHENGHANG, C.; YING, C. Research on reconfigurable manufacturing execution system. In: IEEE. *Intelligent Mechatronics and Automation, 2004. Proceedings. 2004 International Conference on*. [S.l.], 2004. p. 157–161.

CARDOSO, J.; VALETTE, R. *Redes de Petri*. [S.l.]: Editora da UFSC, 1997.

CASSANDRAS, C.; LAFORTUNE, S. *Introduction to Discrete Event Systems*. 2. ed. [S.l.]: Springer, 2008.

CRUZ, D. L. D. et al. Proposta de implementação de controle supervisório em controladores lógicos programáveis. *Anais do IX Simpósio Brasileiro de Automação Inteligente*, p. 19–24, 2009.

CURY, J. E. R. *Teoria de Controle Supervisório de Sistemas a Eventos Discretos*. [S.l.]: V Símpósio Brasileiro de Automação Inteligente, Gramado, 2001.

EBEL, F.; PANY, M. *MPS® Sistema Modular de produção - Estações*. www.festo-didactic.com, Abril 2006.

FENG, L.; WONHAM, W. Tct: A computation tool for supervisory control synthesis. In: *Discrete Event Systems, 2006 8th International Workshop on*. [S.l.: s.n.], 2006. p. 388–389.

GANGNEUX, P. A short term forecasting dss for improving the sales/manufacturing linkage. *Engineering Costs and Production Economics*, Elsevier, v. 17, n. 1, p. 369–375, 1989.

GUIDE, M. U. The mathworks. *Inc., Natick, MA*, v. 5, 1998.

INTERNATIONAL Standard 61131: Programmable Logic Controllers. Part 3: Languages. 1998.

ISA-95. *Enterprise-Control System Integration Part 1: Models and Terminology*. http://isa-95.com/technology-isa95/, 2010. Acesso em 20 de janeiro de 2015.

KLINGE, S. *Supervisory control of a manufacturing cell: modeling and implementation*. Tese (Minor Thesis) — Fakultät für Elektrotechnik und Informationstechnik, Otto-von-Guericke-Universität Magdeburg, 2007.

KOREN, Y. et al. Reconfigurable manufacturing systems. *CIRP Annals-Manufacturing Technology*, Elsevier, v. 48, n. 2, p. 527–540, 1999. <http://www.sciencedirect.com/science/article/pii/S0007850607632326>.

LUSTOSA, L. J.; MESQUITA, M. A. de; OLIVEIRA, R. J. *Planejamento e controle da produção*. [S.l.]: Elsevier Brasil, 2008.

MACIEL, P. R.; LINS, R. D.; CUNHA, P. R. *Introdução às redes de Petri e aplicações*. [S.l.]: UNICAMP-Instituto de Computacao, 1996.

MESQUITA, M. A. d.; CASTRO, R. L. Análise das práticas de planejamento e controle da produção em fornecedores da cadeia automotiva brasileira. *Gestão e Produção*, SCIELO Brasil, v. 15, n. 1, p. 33–42, 2008.

MOLINA, J. R. C. *Uma Abordagem Híbrida para o Controle de Sistemas de Manufatura Baseada na Teoria de Controle Supervisório e nas Redes de Petri Coloridas*. Dissertação (Mestrado) — Pontifícia Universidade Católica do Paraná, 2007.

MORAES, C.; CASTRUCCI, L. *Engenharia de automação industrial*. [S.l.]: Editora LTC, 2007.

NGUYEN, P. et al. *IDES software*. Department of Electrical and Computer Engineering, Queens University, Canada. <https://qshare.queensu.ca/Users01/rudie/www/software.html>.

PINHEIRO, J. *Introdução às Redes de Supervisão e Controle*. Abril 2006. Projeto de Redes. <www.projetoderedes.com.br/artigos>. Acessado em 21 fev. 2013.

PINOTTI, A. J.; LEAL, A. B.; OLIVEIRA, D. S. Uma proposta de metodologia para implementação da estrutura de controle supervisório em controladores lógicos programáveis. *Congresso Brasileiro de Automática (CBA)*, 2010.

PORTILLA, N. B. *Integração de Sistemas SCADA com a Implementação de Controle Supervisório em CLP para Sistemas de Manufatura*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina, 2011.

PORTILLA, N. B.; QUEIROZ, M. H.; CURY, J. E. R. Integration of supervisory control with scada system for a flexible manufacturing cell. *IEEE INTERNATIONAL CONFERENCE ON INDUSTRIAL INFORMATICS (INDIN)*, p. 261–266, July 2014.

QUEIROZ, M. H. *Controle Supervisório Modular e Multitarefa de Sistemas Compostos*. Tese (Doutorado) — Universidade Federal de Santa Catarina, 2004.

QUEIROZ, M. H.; CURY, J. E. R. Modular supervisory control of large scale discrete event systems. In: *In Discrete Event Systems: Analysis and Control. Proc. WODES?00*. [S.l.]: Kluwer Academic, 2000. p. 103–110.

QUEIROZ, M. H.; CURY, J. E. R. Controle supervisório modular de sistemas de manufatura. *Sba: Controle e Automação - Sociedade Brasileira de Automatica*, scielo, v. 13, p. 123 – 133, 08 2002. ISSN 0103-1759.

QUEIROZ, M. H.; CURY, J. E. R. Synthesis and implementation of local modular supervisory control for a manufacturing cell. In: *Discrete Event Systems, 2002. Proceedings. Sixth International Workshop on*. [S.l.: s.n.], 2002. p. 377–382.

RAMADGE, P.; WONHAM, W. The control of discrete event systems. *Proceedings of the IEEE*, v. 77, n. 1, p. 81 –98, jan 1989. ISSN 0018-9219.

RIBEIRO, M. A. *Automação*. 5º. ed. [S.l.], Outono 2005. Tek Treinamento & Consultoria.

ROGERS, G. G.; BOTTACI, L. Modular production systems: a new manufacturing paradigm. *Journal of Intelligent Manufacturing*, Springer, v. 8, n. 2, p. 147–156, 1997. <http://link.springer.com/article/10.1023/A:1018560922013>.

ROSARIO, J. M. *Automação industrial*. [S.l.]: Editora Baraúna, 2009. 515 p.

SEBEM, R.; LEAL, A. B. Escolha entre eventos controláveis em implementação de controle supervisório em clps. *Simpósio Brasileiro de Automação Inteligente (SBAI)*, 2013.

SIEMENS. *TIA Portal: Totally Integrated Automation*. 2012. <http://www.industry.siemens.com.br/automation/br/pt/tia-portal/pages/default.aspx>.

SILVA, A. P. G. da; SALVADOR, M. *O que são sistemas supervisórios?* Dezembro 2011. Elipse Knowledgebase. <kb.elipse.com.br/pt-br/questions/62/O+que+são+sistemas+supervisórios?>.

SILVA, D. B. et al. Dealing with routing in an automated manufacturing cell: a supervisory control theory application. *International Journal of Production Research*, v. 49, n. 16, p. 4979–4998, 2011. <http://dx.doi.org/10.1080/00207543.2010.519732>.

SILVA, Y. G.; QUEIROZ, M. H. Formal synthesis, simulation and automatic code generation of supervisory control for a manufacturing cell. *COBEM, Proceedings of the 20th International Congress of Mechanical Engineering, Gramado, Brazil*, v. 4, p. 418–426, 2009.

SU, R.; WONHAM, W. Supervisor reduction for discrete-event systems. *Discrete Event Dynamic Systems*, Kluwer Academic Publishers, v. 14, p. 31–53, 2004. ISSN 0924-6703.

VALCKENAERS, P.; BRUSSEL, H. V. Holonic manufacturing execution systems. *CIRP Annals-Manufacturing Technology*, Elsevier, v. 54, n. 1, p. 427–432, 2005.

VIEIRA, A. *Método de implementação do controle de sistemas a eventos discretos com aplicação da Teoria de Controle Supervisório*. Tese (Doutorado em Engenharia Elétrica) — Universidade Federal de Santa Catarina, 2007.

WONHAM, W. *Notes on control of discrete-event systems*. 2008. Department of Electrical and Computer Engineering, University of Toronto.

APÊNDICE A – Heurística de Prioridade Definida pelo PCP

```
% % ----- REALIZA CONEXAO
da = opcda('localhost','OPC.SimaticHMI.CoRtHmiRTm');
connect(da);
grp = addgroup(da);
set(grp, 'UpdateRate', .500);

% ----- ESPECIFICA TAGS QUE SERAO LIDAS/ESCRITAS

% -----Tags Modulo de Distribuição

itm0 = additem(grp, 'force_aliment_avanca_Distrib');
itm1 = additem(grp, 'force_aliment_retorna_Distrib');
itm2 = additem(grp, 'force_braco_avanca_Distrib');
itm3 = additem(grp, 'force_braco_recua_Distrib');
itm4 = additem(grp, 'force_vent_pega_Distrib');
itm5 = additem(grp, 'force_vent_ejeta_Distrib');

F = [itm0 itm1 itm2 itm3 itm4 itm5];

itm6 = additem(grp, 'De_aliment_avanca_Distrib');
itm7 = additem(grp, 'De_aliment_retorna_Distrib');
itm8 = additem(grp, 'De_braco_avanca_Distrib');
itm9 = additem(grp, 'De_braco_recua_Distrib');
itm10 = additem(grp, 'De_vent_pega_Distrib');
itm11 = additem(grp, 'De_vent_ejeta_Distrib');
itm12 = additem(grp, 'IP_FI_modulo2_Distrib');
itm13 = additem(grp, 'LED_reset_Distrib');
itm14 = additem(grp, 'LED_Start_Distrib');
itm15 = additem(grp, 'Manual_Mode_Distrib');
%itm16 = additem(grp, 'p_Empurra_Peca_St_Distrib');
%itm17 = additem(grp, 'p_Braco_St_Distrib');
%itm18 = additem(grp, 'p_Liga_vacuo_St_Distrib');
itm19 = additem(grp, 'PecaPosicao_pc_Distrib');
itm20 = additem(grp, 'PecaPosicaoP1_Distrib');
itm21 = additem(grp, 'Reset_Remoto_Distrib');
itm22 = additem(grp, 'running_manual_mode_Distrib');
itm23 = additem(grp, 'Running_Reset_Distrib');
itm24 = additem(grp, 'Running_system_Distrib');
itm25 = additem(grp, 'Start_Remoto_Distrib');
itm26 = additem(grp, 'Stop_Remoto_Distrib');
itm27 = additem(grp, 'unsafe_Distrib');
itm28 = additem(grp, 'vacuoEbraco_Distrib');
itm29 = additem(grp, 'Avanca_alimentador_Distrib');
itm30 = additem(grp, 'Retorna_alimentador_Distrib');
itm31 = additem(grp, 'Avanca_braco_Distrib');
itm32 = additem(grp, 'Retorna_braco_Distrib');
itm33 = additem(grp, 'Pegar_peca_Distrib');
itm34 = additem(grp, 'Ejetar_peca_Distrib');
itm35 = additem(grp, 'activate_router_Distrib');
itm36 = additem(grp, 'LOGCODE_Distrib');
```

```
itm37 = additem(grp, 'Avancou_alimentador_Distrib');
itm38 = additem(grp, 'Retornou_alimentador_Distrib');
itm39 = additem(grp, 'Avancou_braco_Distrib');
itm40 = additem(grp, 'Retornou_braco_Distrib');
itm41 = additem(grp, 'Pegou_peca_Distrib');
itm42 = additem(grp, 'Ejetou_peca_Distrib');
itm43 = additem(grp, 'Chegou_Nova_Peca_Distrib');
itm44 = additem(grp, 'Ocupou_Modulo_seguinte_Distrib');
itm45 = additem(grp, 'Liberou_Modulo_seguinte_Distrib');

itm46 = additem(grp, 'Demanda_Distrib');
itm47 = additem(grp, 'Em_Producao_Distrib');
itm48 = additem(grp, 'Produzidas_Distrib');
itm49 = additem(grp, 'Demanda_pecas_Distrib');

% -----Tags Modulo de Distribuição

itm50 = additem(grp, 'De_ejecao_avanca');
itm51 = additem(grp, 'De_ejecao_recua');
itm52 = additem(grp, 'De_linear_desce');
itm53 = additem(grp, 'De_linear_sobe');
itm54 = additem(grp, 'De_modulo_liberar');
itm55 = additem(grp, 'De_modulo_ocupar');

itm56 = additem(grp, 'force_ejecao_avanca');
itm57 = additem(grp, 'force_ejecao_recua');
itm58 = additem(grp, 'force_linear_desce');
itm59 = additem(grp, 'force_linear_sobe');
itm60 = additem(grp, 'force_modulo_liberar');
itm61 = additem(grp, 'force_modulo_ocupar');

Ft = [itm56 itm57 itm58 itm59 itm60 itm61];

itm62 = additem(grp, 'ejecao_avanca');
itm63 = additem(grp, 'ejecao_recua');
itm64 = additem(grp, 'linear_desce');
itm65 = additem(grp, 'linear_sobe');
itm66 = additem(grp, 'modulo_liberar');
itm67 = additem(grp, 'modulo_ocupar');
itm68 = additem(grp, 'ejecao_avancou');
itm69 = additem(grp, 'ejecao_recuou');
itm70 = additem(grp, 'linear_desceu');
itm71 = additem(grp, 'linear_subiu');
itm72 = additem(grp, 'setor_liberou');
itm73 = additem(grp, 'setor_ocupou');
itm74 = additem(grp, 'proximo_liberou');
itm75 = additem(grp, 'proximo_ocupou');
itm76 = additem(grp, 'peca_disponivel');
itm77 = additem(grp, 'aprovou');
itm78 = additem(grp, 'reprovou');
```

```
itm79 = additem(grp, 'activate_router_Test');
itm80 = additem(grp, 'LOGCODE_Test');
itm81 = additem(grp, 'unsafe_Test');
itm82 = additem(grp, 'Manual_Mode_Test');
itm83 = additem(grp, 'peca_rejeitada');
itm84 = additem(grp, 'pecas_rejeitadas');

log_distrib = [0]         % Inicializa a variável log distrib
log_test = [0]          % Inicializa a variável log teste
TabelaAntiga = zeros(15,1);
SCANTIME = 1.0;
pause(SCANTIME);
seg = zeros(1,15);
seg2 = 0;
segT = 0;


while 1
    if ((itm15.value==1)&&(itm35.value==1)&&(itm27.value==0))        % O sistema deve↙
estar em modo Manual e ativado o botão de Roteamento
        TabelaExcel = xlsread('C:\Users\LAI-↙
Prof\Documents\William\Roteamento\ProgramaMatlab\tabelaPrioridadeComEventosNaoControlavei↙
s.xlsx',1,'D5:D19');
        Pchange = any (TabelaAntiga~=TabelaExcel);
        TabelaAntiga = TabelaExcel;
        if ((log_distrib~=itm36.value) || Pchange)          % O valor deve ser diferente↙
do valor da variálvel LOGCODE
            De = [itm6.value itm7.value itm8.value itm9.value itm10.value itm11.value 0 0↙
0 0 0 0 0 0 0]== 0;
            St = [itm29.value itm30.value itm31.value itm32.value itm33.value itm34.value↙
itm37.value itm38.value itm39.value itm40.value itm41.value itm42.value itm43.value↙
itm44.value itm45.value];
            ACT = (De.*St);        % Verifica quais eventos estão Elegíveis e Habilitados↙
pelo Supervisor

            if any((ACT)~=(seg))

                x_D = itm36.value;
                % ----- DEFINE A SEQUÊNCIA DE PRIORIDADE PARA OS EVENTOS
                for i = 1:15
                    Exec(i) = TabelaExcel(i);
                end
                priority = (Exec.*ACT);

                Y = xlsread('C:\Users\LAI-↙
Prof\Documents\William\Roteamento\ProgramaMatlab\tabelaPrioridadeComEventosNaoControlavei↙
s.xlsx',1,'E5:G5');
                demand = Y(1);
                working = Y(2);
                produced = Y(3);
                % Escreve no SCADA os dados da produção
```

```
                write(itm46,demand);
                write(itm47,working);
                write(itm48,produced);

                if demand == produced         %Alarme no SCADA
                    write(itm49,1);
                else
                    write(itm49,0);
                end
                if (working+produced>=demand)
                    priority(1) = 0;
                end

                m = max(priority); %m recebe o maior valor da tabela de prioridade
                if(m)
                    maximum = find(priority == m); % define a posição do maior valor da↙
tabela
                else
                    maximum = [];
                end

                nmaximum = length(maximum);   % verifica quantos valores máximos tem
                if (nmaximum)
                    r = randi(nmaximum);     %Sorteia entre aqueles que tem valor máximo
                    choice = maximum(r)      %verifica qual a posição na matriz priority↙
foi o sorteado
                    if seg2~=choice
                        if choice<= length(F)    %verifica se a posição sorteada é maior↙
que o tamnaho da matriz F
                            seg2 = choice;
                            seg = ACT;
                            write(F(choice),1);
                            log_distrib(1)=x_D
                            if (choice==1) %Atualiza os dados de peças em trabalho
                                xlswrite('C:\Users\LAI-↙
Prof\Documents\William\Roteamento\ProgramaMatlab\tabelaPrioridadeComEventosNaoControlavei↙
s.xlsx',[working+1],1,'F5:F5');
                                display(sprintf('Demanda: %d\tEm producao: %d\tProduzido:↙
%d\n',demand,working+1,produced));
                                 Y = xlsread('C:\Users\LAI-↙
Prof\Documents\William\Roteamento\ProgramaMatlab\tabelaPrioridadeComEventosNaoControlavei↙
s.xlsx',1,'E5:G5');
                            elseif (choice==6) % Atualiza os dados de peças produzidas
                                xlswrite('C:\Users\LAI-↙
Prof\Documents\William\Roteamento\ProgramaMatlab\tabelaPrioridadeComEventosNaoControlavei↙
s.xlsx',[produced+1],1,'G5:G5');
                                xlswrite('C:\Users\LAI-↙
Prof\Documents\William\Roteamento\ProgramaMatlab\tabelaPrioridadeComEventosNaoControlavei↙
s.xlsx',[working-1],1,'F5:F5');
                                display(sprintf('Demanda: %d\tEm producao: %d\tProduzido:↙
%d\n',demand,working-1,produced+1));
```

```
                                Y = xlsread('C:\Users\LAI-↙
Prof\Documents\William\Roteamento\ProgramaMatlab\tabelaPrioridadeComEventosNaoControlavei↙
s.xlsx',1,'E5:G5');
                            end

                            % Escreva no SCADA os dados da Produção
                            demand = Y(1);
                            working = Y(2);
                            produced = Y(3);
                            write(itm46,demand);
                            write(itm47,working);
                            write(itm48,produced);

                        else
                                display(sprintf('Esperando o evento não controlável %d.↙
\n',choice));   %Se o valor sorteado é maior que o tamanho da tabela F espera terminar o↙
evento não controlável
                        end
                    end
                end
            end
        end
    end

    if ((itm82.value==1)&&(itm79.value==1)&&(itm81.value==0))        % O sistema deve↙
estar em modo Manual e ativado o botão de Roteamento
        TabelaExcelTeste = xlsread('C:\Users\LAI-↙
Prof\Documents\William\Roteamento\ProgramaMatlab\tabelaPrioridadeComEventosNaoControlavei↙
s.xlsx',1,'I5:I21');
        if ((log_test)~=(itm80.value))
            DeTest = [itm50.value itm51.value itm52.value itm53.value itm54.value itm55.↙
value 0 0 0 0 0 0 0 0 0 0]== 0;
            StTest = [itm62.value itm63.value itm64.value itm65.value itm66.value itm67.↙
value itm68.value itm69.value itm70.value itm71.value itm72.value itm73.value itm74.value↙
itm75.value itm76.value itm77.value itm78.value];
            ACTTest = (DeTest.*StTest);        % Verifica quais eventos estão Elegíveis e↙
Habilitados pelo Supervisor

            x_T = itm80.value;

            rejeitadas = xlsread('C:\Users\LAI-↙
Prof\Documents\William\Roteamento\ProgramaMatlab\tabelaPrioridadeComEventosNaoControlavei↙
s.xlsx',1,'F9:F9');

            if itm83.value
                xlswrite('C:\Users\LAI-↙
Prof\Documents\William\Roteamento\ProgramaMatlab\tabelaPrioridadeComEventosNaoControlavei↙
s.xlsx',[produced-1],1,'G5:G5');
                xlswrite('C:\Users\LAI-↙
Prof\Documents\William\Roteamento\ProgramaMatlab\tabelaPrioridadeComEventosNaoControlavei↙
s.xlsx',[rejeitadas+1],1,'F9:F9');
```

```
                write(itm83,0);
            end

            write(itm84,rejeitadas);

            for ii = 1:17
                ExecTest(ii) = TabelaExcelTeste(ii);
            end
            priorityTest = (ExecTest.*ACTTest);

            mT = max(priorityTest); %m recebe o maior valor da tabela de prioridade
            if(mT)
                maximumT = find(priorityTest == mT); % define a posição do maior valor da↙
tabela
            else
                maximumT = [];
            end

            nmaximumT = length(maximumT);   % verifica quantos valores máximos tem
            if (nmaximumT)
                rT = randi(nmaximumT);    %Sorteia entre aqueles que tem valor máximo
                choiceT = maximumT(r)     %verifica qual a posição na matriz priority foi↙
o sorteado
                if segT~=choiceT
                    if choiceT<= length(Ft)    %verifica se a posição sorteada é maior↙
que o tamnaho da matriz F
                        segT = choiceT;
                        write(Ft(choiceT),1);
                        log_test(1)=x_T
                        % Escreva no SCADA os dados da Produção

                    else
                            display(sprintf('Esperando o evento não controlável %d.\n',↙
choiceT));   %Se o valor sorteado é maior que o tamanho da tabela Ft espera terminar o↙
evento não controlável
                    end
                end
            end
        end
    end
end
```